# Extracting Structured Information From Unstructured Text Using Synthesis and Learning

Ajinkya Rajput*
ajinkya@iisc.ac.in
Indian Institute of Science
Bangalore, India

Sahil Garg*
sahilgarg3098@gmail.com
Microsoft Research
Bangalore, India

Sriram Rajamani
sriram@microsoft.com
Microsoft Research
Bangalore, India

Chiranjib Bhattacharyya
chiru@iisc.ac.in
Indian Institute of Science
Bangalore, India

Deepak D'Souza
deepakd@iisc.ac.in
Indian Institute of Science
Bangalore, India

## Abstract

We present a framework to extract structured information from unstructured natural language text. We show that even in an unstructured document we can extract information as long as the information has some syntactic structure to it. We develop a framework that takes a corpus of documents and very few examples of entities to be extracted from these documents. The framework then uses ML and program synthesis techniques to extract the entities from the rest of the documents in the corpus.

***CCS Concepts:*** • **Software and its engineering** → *Programming by example*; • **Information systems** → **Information extraction**.

***Keywords:*** Programming by examples, Information extraction, Text tagging

## 1 Introduction

In this article we tackle the problem of extracting information from unstructured natural language text. We develop a framework that we call BoBW, which stands for Best of Both Worlds. BoBW takes as input a corpus of natural language documents. The user is required to mark entities that she wants to extract in a small number of documents. BoBW then extrapolates from this initial annotation, to annotate similar entities in rest of the documents. The only constraint here is that the entity that the user is extracting should have some syntactic structure. A text has a syntactic structure if it identifies the entity it represents in its form. For example, the text having a structure such as "1080 x 1920", "1080p" and "FHD/HD/UHD" is highly likely to refer to resolution. The complexity of the problem arises from the fact that there can be multiple such syntactic structures for the same entity. Further, we may not know beforehand how many such syntactic structures exist for a given entity in our corpus.

*Both authors contributed equally to this research.

Named Entity Recognition (NER) systems like [5] and [9] works with a fixed set of entities like *Location*, *Organization*, *Person*. They take natural language text as input and assigns entities to tokens in input text. These systems demonstrate impressive results. But these techniques require large amount of data and the entities that these systems identify are fixed with the data-set used for training. A model trained to extract entities from one syntactic structure has difficulty generalizing to a different syntactic structure. We develop a framework that requires minimal human effort of just marking entities in a few (order of tens) documents.

Our work is inspired by [4] which deals with structured attribute extraction from semi-structured text and shows good results on texts of two forms, namely HTML documents and key-value pairs. In this work we do away with the requirement that the document should be semi-structured. Though we use a similar training loop, we identified and implemented significant additions that enable entity extraction to work on unstructured text. These additions are detailed in the rest of this paper.

### 1.1 Approach

We begin by asking the user to annotate the entities she wants to extract in a small subset of the corpus of all documents. We construct a small data set out of these annotations as labels. We expect the size of this data set, and therefore the human effort involved, to be very small.

We then train a Named Entity Recognition (NER) based ML model (details provided in later sections) on this very small data set to obtain a trained model. We run the ML model on all the documents. We also define a field constraint function. This functions allows the user to provide properties of the extracted entities. It is useful in rejecting the outputs that are obviously wrong. For example, you can reject the value "heavy object" when extracting the entity WEIGHT, since we expect presence of digits in the token. We collect those outputs generated by the ML model on the rest of document that are accepted by field constraints and construct a set of tuples where

each tuple contains (*DocumentText*, *OutputValues*). Each of these tuples are input-output examples.

We give this set of input-output examples to a program synthesis module. A string manipulation domain specific language is defined in the program synthesis module. The program synthesis module synthesizes a set of programs that capture the syntactic structure of entities extracted by the ML model. A key observation is that due to poor performance of ML model, occurrences of entities in several documents are not identified by the ML model. In practice, we find that even entities with similar syntactic structure are not consistently identified by the ML model in all the documents. The program synthesis model does exactly that. It captures the syntactic structure of the correctly identified entities and annotates the entities with same syntactic structure in all the documents. These documents where the program synthesizer could automatically annotate and the annotations form another set of input-output examples. This set of input output examples is given back to the ML module to train the ML model. As we show below, with more labelled data, the performance of the ML model improves. This process goes on iteratively as long as we are able successfully extract entities from more and more documents.

### 1.2 Interplay between ML and synthesis

The observation we are leveraging is that the ML model also uses other features other than syntactic structures, and that it can discover entities with new syntactic structures due to the ML model's ability to *generalize*. While this leads to discovery of new syntactic structures, ML models do not consistently identify all entities with same syntactic structure in all documents. Such consistency can be provided by a program synthesizer, by synthesizing program to search for well defined syntactic structure in all documents. The program synthesizer just focuses on one feature i.e. syntactic structure and looks for that features in the whole corpus. In other words program synthesis module does *rigorous application* really well. Here, we are iterating between learning new things and mastering those things. For a good system we need the best of both worlds.

We make the following contribution

- We develop an proof of concept of end to end information extraction framework that can be trained with minimal human efforts that works on unstructured text.
- We develop a string manipulation DSL that is able to prioritize multiple occurrences of a syntactic structure by their relevance to the subject of a document.
- We present a detailed evaluation of our information extraction framework.
- We construct two datasets of natural language text about smart phone specifications and the extracted specifications in english and hindi language.

The rest of the paper is organized as follows, Section 2 provides a motivating example and informally discusses the working of BoBW. In Section 3 we discuss the ML model used. Section 4 sets up the problem formally. We present our proposed methodology in section 5. Section 6 Provides details about the datasets used for evaluation. Section 7 describes the evaluation methodology and presents the results of evaluation. Sections 8, 9 mentions related work and future work respectively. Section 10 provides the conclusions.

## 2 Motivating Example

This framework has the potential of use in a wide variety of applications, that involves extracting certain entities or properties from any corpus comprising of natural language textual data. To provide one such application, consider a corpus of brochures or quotations about valves (in the domain of 'plumbing'). The task at hand is to tabulate multiple specifications of all the different valve types which are described in the brochures. Each brochure contains information on a single valve type. Some example specifications of the valves are valve flux, valve max pressure, diameter etc. Performing this task is of interest because this is typically done by humans so that engineers can make design decisions. Also there are several products other than valves that need information extraction. Some other example use cases can be
-

- Extracting structured information (dates/land dimensions/land areas) from legal documents.
- Extracting runs scored by a player or by a team in different cricket matches from a corpus of blog articles on cricket games (can be generalised to goals scored in a football game/other sports).
- Extracting dates of importance from set of articles on history or politics for creating a timeline or for other further analysis
- Extraction of temperature and weather data statistics from natural language blog articles on a city

These extracted values can then be used for question answering, to develop a dataset for further analysis and even for comparing the articles with regards to structure or information contained in them. The primary aim of this framework is thus, to perform feature and specification extraction for any natural language text corresponding to a fixed given domain and also provide explainability for the technique used to perform the feature extraction so that it can be interpretable as well as generalised to further examples in the domain (which is accomplished by program synthesis).

Our framework works specifically for any independent corpus that may not be popular or easily available, and can be used to extract/highlight non popular entities that questions like "what is maximum operating pressure of rexroth 2FRM6A valve" in the example of the corpus comprising of brochures of valves.

For our evaluation and testing purposes, we are using a data-set of mobile phone review articles and attempting to extract the entity "screen size" from these articles. Although each review article speaks about one mobile phone which is the topic of the article, this phone may be compared with other models or products as well. In such cases, we would like to obtain that value of the screen size that is belonging to the mobile phone which is the topic of the article.

Figure 1. provides a strong example of such a scenario. This is a review article about the phone "Acer Liquid Jade". Having selected a few paragraphs from the article which contain an entity that looks like it could be a screen size (based on the structure learnt for this entity from the outputs of the ML model) in the figure, it can be seen that there are many occurrences of such an entity. These entities are highlighted in yellow, whereas the actual entity that we wish to extract is highlighted in red.

Meanwhile, other key features have been retained starting with the 5-inch display. True, it's not FullHD, but qHD resolutions aren't unheard of in the price segment and 720p gives enough density without rising the price too high. The dual SIM capability will be appreciated by a certain type of users, even if the second card needs to fight with the microSD for the shared slot space.
...
The Acer Liquid Jade S has its contro ls rather evenly distributed. The top right (when viewed from the front) is the home to the power button. It's a location which may have been acceptable on the 4-inch iPhones, with their dedicated home button in closer reach, but is very much out of place on a 5-inch smartphone where it's the only means to wake up the device.
Moving on, towards the left of the top you'll find the 3.5mm headphone jack, while in between resides the pinhole for the secondary mic. The bottom is home to the microUSB port on the right and the primary microphone is even closer to the corner.
...
Display
**The Acer Liquid Jade** is equipped with a 5-inch IPS LCD display with a 1,280 x 720 pixel resolution. Stretched over this diagonal, that results in the decent 294ppi density. It's not class-leading numbers and competitors can be found, which sport FullHD panels. Then again, the hugely popular

**Figure 1.** Extraction of screen size entity from review article on "Acer Liquid Jade"

In the second paragraph, there is a mention of the screen size being 5 inch, but this is not directly related to the subject and is instead mentioned as a feature that was retained from the previous version of the phone "Acer Liquid Jade". The third paragraph again contains entities that look like screen size, but this is because it is comparing another feature for different screen sizes and is not actually indicating the screen size of the subject. In the last paragraph, highlighted in red, is the correct screen size value of the phone which is the subject, and this is the value that we wish to extract using our framework. When the ML model runs on this article, it returns the value of "5-inch" for the annotation but it cannot be interpreted why the ML model picked this value.

There are also multiple instances where the ML model generated the wrong output, that was not the screen size of the subject of the article. An example of this is provided in Figure 2, which is a review article on the phone "Google Pixel". The value highlighted in red is the one we wish to extract and is corresponding to the screen size of Google Pixel, the value highlighted in green is the one that is obtained as output from the ML model and the yellow-highlighted value is another occurrence that follows the syntactical structure of the entity and could have been picked up by the ML model but does not relate to the subject of the article.

Display

The Google Pixel rocks a 5-inch Samsung AMOLED screen with 1080p FullHD resolution and it carries the same pixel density as the 4.85-inch LG Nexus 5 did back in 2013 with a ppi of 441. For a phone in late-2016, many would expect it to have a QHD screen by default but that's perhaps an overkill on a 5-inch screen.

The AMOLED display is gorgeous and colors, though vibrant, don't look too oversaturated. The screen sits behind a panel of 2.5D Gorilla Glass 4, which subtly curves into the screen's bezel.

**Figure 2.** Extraction of screen size entity from review article on "Google Nexus"

In both of these aforementioned examples, it is seen that the ML Model does output a value for the entity to be extracted but it does not use any knowledge regarding the subject or domain of the articles and could generate both correct outputs (as in the former case) or wrong outputs (as in the latter case). Finally, there can also be another third case, where the ML model does not generate any output at all but the value of the entity does exist in the article. An example of this is shown in Figure 3, where the text highlighted in red is the value to be extracted and is found when the extraction on the article is attempted using program synthesis. The ML model did not generate any annotation for this review article on "Oppo F7". Through our framework, the mistakes made by the ML model or the values that were missed out could be extracted with the help of program synthesis. The programs generated would also be interpretable and easy to read and understand. The programs for these articles that are learnt are as below -

- $SubstringNearSubject(v, RelPos(v, (s, ([0-9]1, 2 .[0-9]1, 2.0, 3) - inch)), RelPos(v, (([0-9]1, 2 .[0-9]1, 2.0, 3) - inch s)))$
- $SubstringNearSubject(v, RelPos(v, (s, ([0-9]1, 2 .[0-9]1, 2.0, 3) - inch)), RelPos(v, (([0-9]1, 2 .[0-9]1, 2.0, 3) - inch, s)))$
- $SubstringNearSubject(v, RelPos(v, (s, ([0-9]1, 2 .[0-9]1, 2.0, 3)inch)), RelPos(v, (([0-9]1, 2 .[0-9]1, 2.0, 3)inch, [, .?!])))$

## 3 The ML Model

For the ML model, we use the Spacy[1] library. It is a library for natural language processing built with objective of getting tasks done quickly. It provides a Tokenizer, Named Entity Recognizer(NER), Part of Speech tagger and a dependency parser. Spacy also provides pre-trained language models.

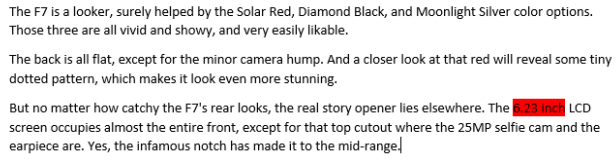


**Figure 3.** Extraction of screen size entity from review article on "Oppo F7"

For our ML model, we first train the NER in spacy. To train the NER, spacy requires user to provide annotation in the form of $[start\_index, end\_index, ENTITY\_NAME]$. We provide these annotations from a human user in the first iteration, and then the annotations are generated by the program synthesis module in subsequent iterations which are fed to Spacy in the training procedure.

Once NER is trained, it identifies the entities that we defined in the documents provided to the model.

## 4 Formal description of the problem

The problem that we are trying to tackle is of extracting heterogeneous entities from natural language text. In this section, we formalize this problem and present the outline of the Entity-extraction framework that is used. Here we provide some definitions and set terminology to aid formal description of the problem.

**Data-sets and Fields**. The input to the framework is a tuple (D, E) where D is a set of natural language documents and E is a entity. Entities are the name of information that is to be extracted from the document. All the documents in D are assumed to be on the same topic. Examples of data sets and entities are a collection of articles about various smartphones with SCREEN_SIZE as entity to be extracted, a collection of various commentaries of a sport with DATE of the match as the entity etc. An additional constraint on entities is that the values for those entities in the documents should have some syntactic structure. For example the entity DATE has syntactic structure like dd-mm-yy. Note that there can be multiple syntactic structures for the same entity and this syntactic structure need not be same for the entities in all the documents, for example dates can be given in following format dd-mm-yy(12-10-19), dd-mm-yyyy(12-10-2019), dd-mmm-yy(12-Oct-2019), dd/mm/yyyy(12/10/2019).

For the entity $E$ we define a partial function, field $f_E : D \nrightarrow < int, int >$. The field takes in a document and returns a substring of the document that is the value of the entity in the input document. The returned stubstring is represented by a tuple of integers, starting index and ending index in the input document respectively. The field function is a partial function because the field function $f_E$ may not return anything ($\perp$) if entity is not present in the document.

In current setting of the problem, we consider only one entity to be extracted. We do so only for simplicity of presentation. However, the framework can be used to extract multiple entities just by running the framework once for each entity. The problem can also be set up to handle multiple entities together without much effort.

**Field Constraints**. Given a field $f$ on a data-set $D$, a field constraint $C_{D,f} : String \rightarrow \{true, false\}$ is a constraint that is dependent on the document and the prospective values of the respective field. Its a mechanism for user to provide the properties that can out rightly identify wrong outputs. Ideally, a constraint would have $C_f(i, o) = true$ if and only if $o = f(i)$. In practice, such constraints can be difficult to write or specify, since the user may not have a deep understanding of what the field values look like, and so, we provide a model where soft constraints can be given that are close approximations to the actual constraints. It should be noted here that the system can accustom to noise constraints which are either an over-estimation or an over-approximation. These constraints can be set for particular input documents or for the entire data-set corresponding to a field. Field constraints do not necessarily impose structure on the problem, these function can be really trivial for human to come up with. Example of field constraints for entity SCREEN_RESOLUTION is that the output must have some numeric characters in them.

Thus, the extracted field which is the output is a sub-string of the input, which follows the given constraints for the field (if any) such that no extension of the output satisfies the conditions completely.

**Annotations**. Given an entity $E$ with corresponding field $f_E$ for a data- set D, an annotation, annotation(d), of a document $d \in D$ is a $(o, e)$ where: (a) $o = f_E(d)$ are the ground-truth outputs (outputs of field functions defined in paragraph above), and (b) $e$ is some optional supplementary information given relevant to the field. In our implementation we use $e$ to provide name of the product of the article to synthesizer so that programs can prioritize entities with similar syntactic structure.

**Specification**. Specification is a set of input output examples provided to program synthesizer. Specification is defined as $\{(d, s) | d \in D' \wedge$ s is substring of d$\}$, where $D'$ is some subset of set of articles, $D$. Specification is also referred as Spec. A program $P$ is said to satisfy a specification $S$ if $\forall (a, o) \in S, P(a) = o$

**SoftSpec**. Sometimes it might be impossible for a single program to satisfy a specification. SoftSpec is set of input output examples which can be partitioned into multiple sets of input output examples such that each partition is a specification and therefore can be satisfied by one program each.

### 4.1 The Entity-Extraction Problem

We can now formally define the entity-extraction problem. The extraction framework is provided with following inputs:

1. A corpus of documents $D$ from a data set.
2. An entity that is to be extracted $E$.
3. an subset of inputs to be initially annotated using oracle $D_{init} \in D$. i.e. seed inputs.
4. an optional field constraint $C_{D,f} : String \rightarrow \{true, false\}$

For the above mentioned inputs, the entity extraction problem is to find a function $f_E^*$ that is approximates $f_E$

## 5 The Entity-Extraction Framework

The meta algorithm for our entity-extraction framework is shown in Algorithm 1. In addition to the inputs $D, D_{tr}, C_{D,f}$ described above, we supply to it a Domain Specific Language $L$. Figure 4. shows the overall architecture of our entity-extraction framework. We briefly explain each of the components of the framework here; full explanations follow in the subsequent paragraphs.

**Algorithm 1:** Outer loop of entity extraction framework

**Require:** Set of all the articles $D$
**Require:** Set of oracle annotated articles $D_{tr}$
**Require:** Field constraint $C_{D,f} : String \rightarrow true, false$
**Require:** Annotation oracle $AOracle$
**Require:** DSL $L$
**Result:** Trained model $M$

$Annots \leftarrow \{(i, AOracle(i)) | i \in Dtr\}$;
**do**
  $M \leftarrow MLTraining(annots)$ ;
  $SoftSpec \leftarrow \{(i, M(i)) | i \in D\}$;
  $R \leftarrow LearnRegex(SoftSpec)$;
  $P \leftarrow ProgSynthesiser(D, SoftSpec, L, C_{D,f}, R)$;
  $NewInputs \leftarrow D/D_{tr}$;
  $(D_{succ}, NewAnnots) \leftarrow RunOnNewInputs(NewInputs, P, M)$;
  $Annots \leftarrow Annots \cup NewAnnots$;
  $D_{tr} \leftarrow D_{tr} \cup D/D_{succ}$;
**while** *stopping_condition*;
return $M$;

**MLTraining**. The MLTraining procedure takes a set of annotated documents as input and it generates a ML model $M$. The ML models that we are using for this training are described in Section 3. Given any input $d \in D$ and the ML model $M$, $M(d)$ returns a substring of $d$. Ideally, $M(d)$ is the entity we wish to extract. The principal advantage of using ML models, intuitively, is that they generalize across different syntactic structure. ML models leverage a high level features like shape of the entity, words following and before the entity, vector representation of words etc. The model training procedure we use is detailed in Section 3.

**SoftSpec**. Once the ML model, M, is trained we run the model on all the inputs in $D \backslash D_{tr}$. The set of tuples $\{(d, M(d) | d \in D \setminus D_{tr}\}$ forms a soft specification labeled SoftSpec.

**LearnRegex**. The LearnRegex procedure takes SoftSpec as input and returns a list of regular expressions that may prove useful in synthesizing programs. The details of this procedure are presented in Section 5.2.

**The ProgSynthesiser**. The ProgSynthesiser procedure is provided SoftSpec, along with any field constraints the user has provided. This procedure is also provided the set of regular expressions R, which are learned by the system using the model outputs. The ProgSynthesiser procedure returns a set of programs $P = \{P_0, P_1, \ldots, P_m\}$. We call this sequence of programs a disjunctive program. For a document $d$, we have that $P(d)$ returns $P_i(d)$ for least $i$ such that $P_i(d)$ satisfies the field constraint. Ideally each program captures one syntactic structure. We seek to generate $P_i$ such that it either generates the correct output if the document contains the entity with the syntactic structure that $P_i$ captures, or outputs null otherwise. The disjunctive program is thus able to span the entire set of syntactic structures that are captured by each of the programs $P_0, P_1, \ldots, P_m$. As mentioned in subsequent sections, the synthesized disjunctive program $P$ is able to enhance the precision and recall by a significant factor compared to the machine learning model $M$ in isolation. In addition, the programs are interpretable and can be easily debugged. The disjunctive synthesis procedure is detailed in Section 5.2.

**RunOnNewInputs**. The RunOnNewInputs procedure uses these synthesized disjunctive program $P$, obtained from the ProgramSynthesiser procedure. It runs $P$ on all articles except those already already passed to synthesizer i.e., $D \setminus D_{tr}$.

The above 4 steps are repeated until some stopping condition is reached. This stopping condition can be fixed number of iterations. Another stopping condition can be a plateau in performance measures etc. In our implementation,, we fix the number of iterations (usually a small number 3-5).

### 5.1 Program Synthesis from SoftSpec

As shown in Algorithm 1, the ProgSynthesiser step takes as input a) set of articles $D$, b) a soft specification SoftSpec c) a domain specific language $L$, d) an optional field constraint function $C_f : D \times String \rightarrow \{true, false\}$, and e) a list of regular expressions $R$. The goal of the disjunctive synthesis step is to produce a disjunctive program that can extract output field values.

The output of ML model is noisy because the ML model considers a lot of features simultaneously, and in our setting it has a small amount of initial training data. Because of this ML model correctly identifies the chosen entities in only very few documents. There is also a high false positive rate, meaning ML model outputs substrings which

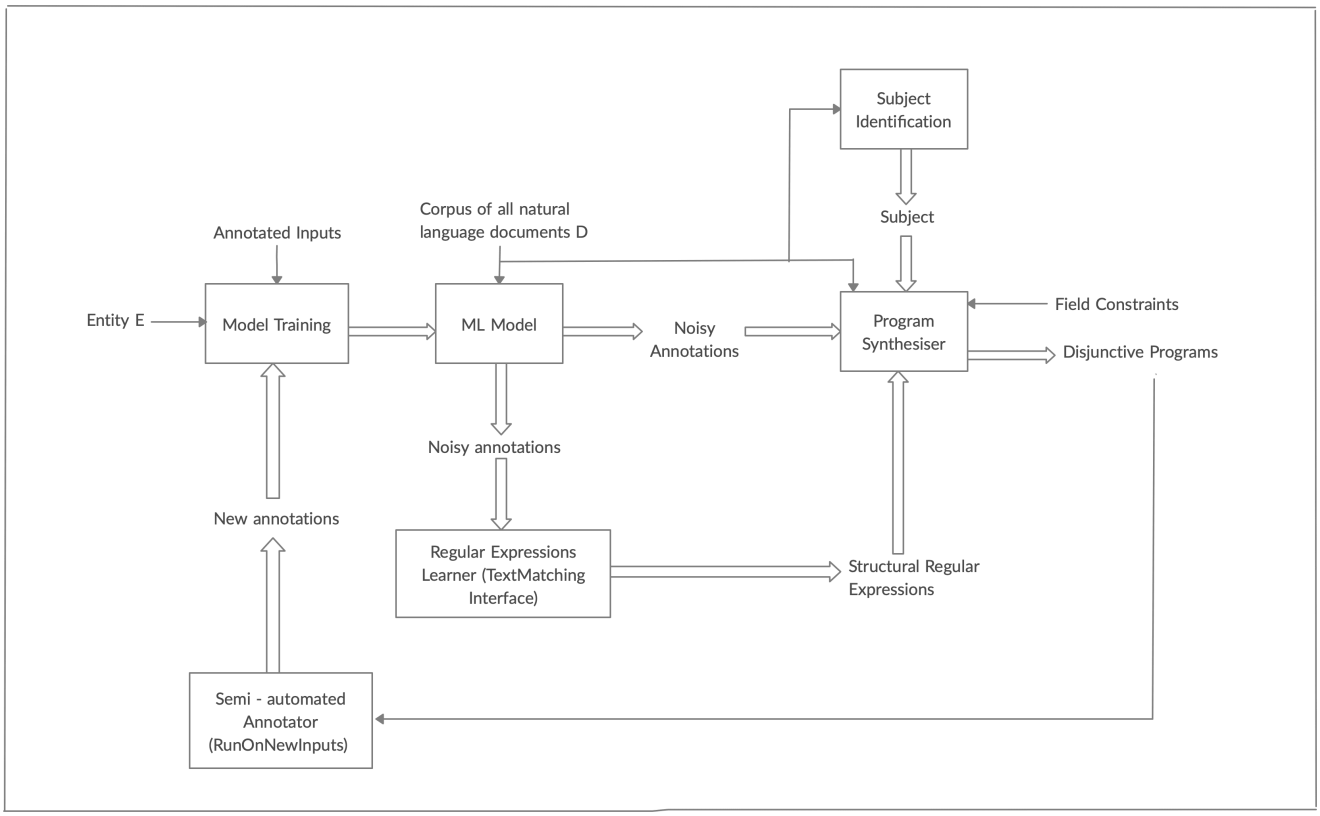


**Figure 4.** Block diagram of the proposed Entity Extraction Framework

are not entities. However, even these few correctly identified entities implicitly provide instances of syntactic structures of the entity. For example, consider an entity DATE, if 12/10/2019 is identified as correct output, we know that dd/mm/yyyy is one of the possible syntactic structure of the entity DATE_OF_PUBLICATION. However, as the initial ML model is inaccurate, it misses entities with same syntactic structure in other documents. By synthesizing a disjunctive program we aim to leverage the syntactic structure that we have discovered and annotate other entities that have the same syntactic structure in other documents. With carefully designed DSL we can make sure that we do not annotate occurrences which conform to syntactic structure but are not the entities that we want to extract. For example other dates in articles which are not date of publications.

### 5.2 Disjunctive Program Synthesis Algorithm

In this section we give details about the program synthesis module. This module is denoted by ProgSynthesizer in Algorithm 1. We first setup some terminology and building blocks. Then describe workings of the algorithm.

**Example-based Synthesis**. We use Microsoft PROSE framework of program synthesis[6]. PROSE lets us provide a grammar and semantics of Domain Specific Language (DSL) and synthesizes programs in the provided DSL. Prose takes a specification as input and returns program that satisfies the specification. This is denoted by function $Synth$ in Algorithm 2

**Disjunctive Programs**. A disjunctive program is a sequence of programs that satisfy a SoftSpec. The sequence of programs is constructed with the objective that the these programs cover all the syntactic structures that occur in all the documents. We call this sequence of programs A disjunctive program $P$ is defined as a sequence of programs $\{P_0, ..., P_m\}$ where each program $P_j : D \not\to String$ is a partial function such that, for a document $d$, $P(d)$ outputs $P_j(d)$ for least $j$ such that (a) $P_j(d)$ is defined and (b) $P_j(d)$ satisfies the field constraint. Please note that the disjunctive program is a sequence and not a set. It is easier to explain reason in context of explanation of Algorithm 2 in subsequent paragraph

**Synthesis Objectives**. Input to the algorithm is SoftSpec(set of input output examples). The objective of the algorithm is to builds clusters(smaller sets) of these input examples such that each cluster can be satisfied by a single program. Thus the SoftSpec is partitioned into so called HardSpecs. We want to build these clusters(hardspec partitions) in a way that maximizes the agreement of the synthesis and the ML model.

**Algorithm 2:** Algorithm to generate disjunctive program

```
Require: SoftSpec SS
Result: Sequence of programs
1  programs ← [] ;
2  ioClusters ← [] ;
3  UnlearnableIO ← [] ;
4  clusterToProg ← {};
5  for (article, op) ∈ SS do
6      specIncorporated ← False;
7      for cluster ∈ ioClusters do
8          prog ← clusterToProg(cluster) ;
9          if prog(article) = op then
10             clusterToProg.remove(cluster);
11             cluster ← cluster ∪ {(article, op)};
12             clusterToProg.add(cluster, prog);
13             specIncorporated ← True;
14     for cluster ∈ ioClusters do
15         if specIncorporated then
16             break;
17         tempCluster ← cluster ∪ {(article, op)} ;
18         prog ← Synth(tempCluster);
19         if prog ≠ null then
20             clusterToProg.remove(cluster);
21             cluster.add((article, op));
22             clusterToProg.add(newCluster, prog);
23             specIncorporated ← True ;
24             break ;
25     if !specIncorporated then
26         newCluster ← new Cluster((article, op));
27         prog ← Synth(newCluster) ;
28         if prog ≠ null then
29             ioClusters.append(newCluster);
30         else
31             UnlearnableIO.add((article, op));
32     sortClusterDecendingBySize(ioClusters);
33 for cluster ∈ ioClusters do
34     programs.append(progToCluster(cluster));
35 return programs
```

With the definition of disjunctive programs and synthesis objectives setup, we present the details of the synthesis algorithm.

**Disjunctive program synthesis algorithm**: Even with a powerful framework like PROSE at hand, the synthesis is challenging because we are trying to learn multiple syntactic structures and a single program that covers all the syntactic structure might not exist with a simple DSL described above. One obvious choice is to make the DSL more expressive but that will increase the complexity of the DSL implementation and it won't be easy to substitute DSLs while applying the framework in different domains. We take an alternative approach in which we try to synthesize multiple simple programs that each captures only one or few syntactic structures. As defined before we call this set of programs a disjunctive program. To synthesize disjunctive program we need to cluster the input output examples into different clusters and try to synthesize a program using PROSE for each of these clusters. The strategy for forming these clusters is given in Algorithm 2.

Data structures and helper routines used in Algorithm 2

- **programs**: A list of programs
- **ioCluster**: A set of input output examples.
- **ioClusters**: A list of specifications. Note that each specification is a set of input output examples. Therefore ioClusters is a list of sets of input output examples.
- **unlearnableIO**: A list of input output example for which prose could not generate any progrma
- **clusterToProgram**: A map from specification i.e set of input output examples to program. Therefore we store a program corresponding to each cluster.
- **Synth**: A call to prose framework to generate a program for a specification
- **sortClusterDescendingBySize**: A helper routine to sort the list ioClusters in descending order of size of its clusters.

Remember, the objective of the algorithm is to partition the SoftSpec into clusters.

The outer for loop on line 5, iterates over input output examples i.e. $((article, op))$ in SoftSpec $SS$. $specIncorporated$ is a flag that keeps track of weather current input output example is added to some cluster. First this flag is set to $False$.

Then, first inner for loop (lines 7-13), iterates through already built clusters. For each cluster, it executes the program corresponding to that cluster with $article$ as input. If the output of the program is equal to $op$, the program is added to that cluster and map $clusterToProg$ is adjusted. And we move on to next input output example

If all currently synthesized programs are unable to generate correct output $op$ then we try to synthesize new programs (lines 14-22). For each cluster we create a $tempCluster$ by adding current input output example to current cluster. We try to synthesize a program on $tempCluster$. If a such a program is possible it will be synthesized. If program is synthesize the mapping $clusterToProg$ is adjusted and we move on to next input output example.

If $Synth$ was unable to generate programs for any of the clusters along with current input output example, We try to learn a program only on current input output example (lines 25-29). We create a $tempCluster$ with only current input

output example in it and we try to learn a program for this. If a program is synthesized for this we store *tempCluster* in *ioClusters* list

Else we add current input output example to $UnlearnableIO$ list

### 5.3 Importance of sorting

After processing each input output example this list is sorted according to cluster size. This is important because when a cluster size is big that means the there exists a program by construction which satisfies all the input output examples in that cluster. Even though the noisy ML model generates a lot of outputs which are not the expected entity, we leverage the following facts

- All the correctly identified entities, however less in number have a syntactic structure, by construction.
- All the incorrect strings output by ML model will not have structures in common. We assume that ML model is slightly skewed toward the correct entities therefore it is not consistent in outputs of incorrect entities.

Therefore the cluster for correct entities is slightly bigger in size and captures the correct syntactic structure. When running the disjunctive program on rest of the articles for generating new annotations we wish to give prefer extracting articles according to program that is more likely to capture the syntactic structure of expected output.

**Entity Extraction DSL L :** The syntax for synthesis programs is given by grammar in Figure 5. The DSL is a inspired from the FlashFill DSL[2] that uses strings and regular expressions. Following are some of the important features of the grammar

- The @input construct: Here, @input represents the placeholder symbol which holds inputs given to programs constructed in DSL. Here the input to programs constructed in this DSL is a tuple of two strings, the first being the text of the document and the second specifying the subject of the article. In case no subject is found for the article, then this element of the tuple is null. The subject is not the input given to the our framework but is computed by the framework as a preprocessing steps.
- Operator SubstringNearSubject(v, pos, pos) and Substring(v, pos, pos): Both these operators take a input string, v, and two positions, i.e. pos, pos. And returns a substring in v from first pos till second pos. SubstringNearSubject(v,pos,pos) also does the same with a small difference, it returns the substring demarcated by pos, pos if subject, provided in second string of 2-tuple input is present within look around window of a threshold from start.
- Operator RelPos(v, rr): This operator lets us search the input string using regular expressions. This operator takes a string, v, and a 2-tuple of regular expression and returns an integer index, say i, in v such that at the first regular expression matches on left of i and second expression matches on right of i. For example, substring("ab98ef",([a-z]+, d+)) will return index 1 because at index 1, ab matches [a-z]+ on left and 98 matches d+ on right.

```
@input Tuple<string,string> v;

program :=  SubstringNearSubject(v, pos, pos)
 | Substring(v, pos, pos);

pos := RelPos(v, rr);

int k;
Tuple<Regex,Regex> rr;
```

**Figure 5.** Syntax of the substring-based DSL language L

**Preprocessing**. We perform two preprocessing steps in each iteration after ML model produces noisy outputs and before calling synthesizer.

1. Learning Regex: The RelPos(v, rr) operator take as input a tuple of operators. While synthesizing programs, the framework tries combinations of couple of regular expressions from a list of regular expressions. This step builds that list of regular expressions before the program synthesis begins. To build the regular expressions we use the TextMatching interface provided by PROSE. This API takes a collection of string and returns a collection of regular expressions that cover the input list of strings. The text matching API performed poorly when we provided all the outputs of ML model to TextMatching API. To get a list of useful regular expressions, we randomly sample the outputs from ML model and call TextMatching API on that sample. We repeat this till all the string in outputs are included in a sample at least once.
2. Subject Identification: For each article we try to identify the subject of the article. We do this using a regular expression [A-Z][a-zA-Z0-9]* [a-zA-Z0-9]+ that represent the subject of the conversation. We leverage a basic assumption that the subject of the article will occur with highest frequency. In our experiments we see that it works pretty well. For instance, in our implementation for the data-set of mobile phone reviews, the subject for a review article on a Samsung Galaxy S5 phone was obtained as "Samsung Galaxy S5" although there were instances of many other names of phones in the article such as Iphone 5 and Motorola, because the frequency count was highest for Samsung.

# 6 Datasets

-What dataset are we using We perform our experiments on two datasets.

- Smartphone blog articles, $D_b$
- Hindi news articles about gadgets, $D_h$

## 6.1 Smartphone blog articles, $D_b$

The dataset $D_b$, is a collection of 1100 blog articles about smartphones in english langugae. Most articles are about one smartphone each. These articles are intended to inform buyers about specifications, pros and cons and comparisons with similar smartphones. The dataset also contains articles which are about technology events and few comparison articles about more than one smartphones. All articles do not follow any presentation template. The length of the article varies from min 200 words to 20389 words.

## 6.2 Hindi news articles about gadgets, $D_h$

The dataset $D_h$, is a collection of 526 technical news articles collected from website of news channels. The news articles are from "smartphone and laptop news" in "technology" section of the news website. The length of these articles vary from 44 words to 3814 words with mean length of around 300 words.

# 7 Evaluation

## 7.1 Evaluation Methodology

For both the data sets we, partition the dataset into test_set and train_set. test_set is generated by random selection. The size of test set is approximately 10% of the size of the dataset. Only the train_set is used during training the ML model and synthesizing the programs. For the articles in test_set we have manually extracted entities. As seen in previous sections, the proposed technique proceeds in iterations of synthesizing programs and training the ML model. After each iteration we get a trained ML model. We use the trained ML model to extract entities from articles in test_set. Calculate precision, recall and F1 score, by considering the manually extracted entities as ground truth. The metrics are defined as follows,

For a selected entity, Let

$M$ be the trained model,

$G$ the ground truth tabulated by manually going through test_set,

$a \in test\_set$ be articles in test_set,

$M(a)$ and $G(a)$ be the entity extracted by the model and manually respectively, and

$\perp$ denotes when model or ground truth does not return any value.

- Precision p: This is the standard measure of false positives used in information retrieval, and is defined as:

$$p = \frac{|\{a \in test\_set | M(a) = G(a) \wedge M(a) \neq \perp|}{|\{a \in test\_set | M(a) \neq \perp\}|}$$

- Recall r: This is the standard measure of false negatives used in information retrieval, and is defined as:

$$p = \frac{|\{a \in test\_set | M(a) = G(a) \wedge M(a) \neq \perp|}{|\{a \in test\_set | G(a) \neq \perp\}|}$$

- F1 score: This measure combines precision and recall into a single number, and is computed as 2pr /(p + r ).

## 7.2 Experimental Setup:

We train a Named Entity Recognition (NER) model using residual convolutional neural networks with the help of the standard implementation of a tool named “Spacy”[1]. For program synthesis, we use Microsoft "PROgram Synthesis using Examples" (PROSE) library [6]. We use the DSL shown in figure 5. All experiments below are performed on screen size entity because it is present in all articles so ample ground truth is available. It is present in different syntactic structures. It is present in both Hindi and English datasets as English datasets is only about smartphones where as hindi dataset has articles about other gadgets like laptops etc. Also screen size is easier to check with ground truth.

## 7.3 Evaluation

We wish to answer the following research questions -

- Q1 : Does combining ML model with the program synthesis help in improving the model performance?
- Q2 : Does performing multiple iterations between the ML model and program synthesiser result in better extraction?
- Q3: When information related to the subject of the article is added to the DSL, does it lead to better programs for purpose of extraction?
- Q4 : Does this method generalize well?

Due to the stochastic nature of the ML model and the program synthesiser, the framework does not provide any formal guarantees regarding the extracted entity, but aims to provide an answer as close as possible to it.

## 7.4 Does combining ML model with the program synthesis help in improving the model performance?

To answer this question we compared the performance of model trained by proposed hybrid approach with 12 annotations against spacy NER model trained using the same annotations used for hybrid approach. Additionally we also trained spacy on 1, 5, 11,16 annotations. The entity to be extracted is screen size. The results are presented in Table 1. One point to note here is that recall is very low for only ML methods where as recall is significantly more for hybrid

| Config | Prec | Recall | F1 |
|---|---|---|---|
| Hybrid | .83 | .72 | .77 |
| OnlyML-hyb | .90 | .55 | .67 |
| OnlyML-1 | 0 | 0 | 0 |
| OnlyML-5 | .9318 | .41 | .57 |
| OnlyML-11 | .57 | .04 | .075 |
| OnlyML-16 | .7 | .12 | .2 |

**Table 1.** Comparison with hybrid training with pure ML training

approach. This result shows that hybrid approach improves the performance of the system overall.

### 7.5 Q2 : Does performing multiple iterations between the ML model and program synthesiser result in better extraction?

To answer this question we compare performance metrics after each iteration of proposed approach. The entity to be extracted is screen size

| Iteration | Prec | Recall | F1 |
|---|---|---|---|
| 0 | .85 | .59 | .70 |
| 1 | .70 | .60 | .65 |
| 2 | .79 | .68 | .73 |
| 3 | .76 | .68 | .71 |
| 4 | .78 | .72 | .75 |

**Table 2.** Effect of multiple iterations

We see similar trends in other experiments also. The trend shows that multiple iterations between ML model and synthesis improves system performance. One thing to note is recall increases with number of iterations.

### 7.6 When information related to the subject of the article is added to the DSL, does it lead to better programs for purpose of extraction?

To get insight in this matter we compare the performance of system with and without SubstringNearSubject operator in the DSL. We present the metrics in Table 3 Columns with with suffix "w/sub" are results of DSL with SubstringNearSubject operator and columns without the suffix are results of DSL without that operator.

It can be noted that metrics for DSL with SearchNearSubstring operator increase faster than that of a DSL without the said operator. The precision and recall of DSL with the operator is higher than the DSL without operator.

### 7.7 Q4 : Does this method generalize well?

To answer this question we repeated the experiment to extract screen size on dataset $D_h$. This dataset is in hindi. The proposed approach with only one change. SubstringNearSubject operator was removed from DSL. The approach performed well with virtually no change at all. In Table 4 we present iteration wise metrics. The entity to be extracted is screen size.

Here we can see that initially with even with 13 annotation the precision, recall and F1 score are zero. In comparison on English dataset the F1 score is between .55 and .60. We believe this is because the language model for the Hindi in spacy is not as mature as English. We see steady rise in all 3 metrics with iterations. This result shows promise that proposed approach can be used to speed up training NLP models in non english language where, language and training data is scarce.

## 8 Related Work

In machine learning literature, this work falls in broad area of Question Answering. The Question Answering literature is further divided into question parsing and understanding, information extraction, and answer generation. This work is closely related to information extraction. Information extraction is popular and widely studied field. [7] presents a popular framework for automatically learning rules or information extraction system and works on both free and semi structured text. More recently [10] employs bidirectional transformers, a new deep neural network architecture for information extraction task. It tries to derive quantitative entities from business documents. [9] deals with extracting information from semi structured text using probabilistic parsing of a discriminative grammar. They demonstrate their technique by extracting individual fields like first name, last name, home phone number etc. from an address.

Another area that is closely related to present work is that of Named Entity Recognition. It is usually used inside information extraction systems to extract entities and leverage their relations with other entities for higher level reasoning for information extraction. Some of state of art in named entity recognition is [8]. They present a new architecture for neural network which they call Iterative Dilated CNNs. This is an improvement in terms of training time and performance of entity recognizer in [5]. [5] provides two models i.e. LSTM based and CRF based. They show good performance on dataset in English, Dutch, German and Spanish. Both these works are tested on CoNLL dataset[3] which have only ORG, PERSON, LOC and MISC entities.

Our work is closely related to [4] which performs the task of entity extraction from semistructured datasets. They use 2 datasets namely political dataset which is a store of key value pairs, and a machine to human emails dataset which are sent by travel aggregators, and had travel details of the recipents of email. In the first case they extract entites selected by user and in second case they extract the specific details about the travel from the email e.g boarding gate, time

| Iteration | Prec | Prec w/sub | Recall | Recall w/ sub | F1 | F1 w/ sub |
|---|---|---|---|---|---|---|
| 0 | .85 | .90 | .59 | .55 | .70 | .68 |
| 1 | .70 | .83 | .60 | .72 | .65 | 77 |
| 2 | .79 | .83 | .68 | .70 | .73 | .76 |
| 3 | .75 | .79 | .68 | .69 | .71 | .74 |
| 4 | .78 | .78 | .72 | .68 | .75 | .73 |

**Table 3.** Effect of SubstringNearSubject operator

| Iteration | Prec | Recall | F1 |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 1 | .78 | .40 | .53 |
| 2 | .78 | .55 | .65 |
| 3 | .9 | .66 | .76 |
| 4 | .94 | .66 | .78 |

**Table 4.** Performance on Hindi dataset

of departure etc. This work also synthesizes a disjunctive program and proposes an approx max cover algorithm, which is a little different from what we do. One major difference in approaches is that the proposed work here uses structure in the information extracted while [4] uses the semi-structure of the dataset.

## 9 Future Work

1. Preliminary results show promise in the proposed approach but we really want to stress test the system. To see what breaks the system? What kind of inputs break the assumptions stated in section 5.3 about importance of sorting?
2. We started with substring language used in flashfill. The current DSL operates at granularity of characters. Since we need to anyway extract substrings from the articles we should work at a granularity of tokens. The token based DSL should be designed from scratch.
3. Proving formally that minor skew in the results of ML model can be leveraged to train the ML model better. If this is not possible, formulating the minimum requirements on input annotations for this system to work with high probability.

## 10 Conclusion

We developed a framework which uses machine learning models and program synthesis for entity extraction from unstructured data. The proposed framework works with very few seed examples. We leverage the structure of information that is to be extracted. We show improvement in performance of whole system by carefully designing DSL using a feature used in classical ML system for natural language processing thus showing potential in innovative DSL designs. We are going to undertake this task as part of future work. We also provide a argument about how ML and program synthesis models complement each other to utilize the information captured by each other. We present results of a preliminary evaluation which shows promise and motivates further rigourous investigation.

## References


[1] [n.d.]. spaCy, Industrial-strength Natural Language Processing in Python. https://spacy.io/

[2] Sumit Gulwani. 2011. Automating string processing in spreadsheets using input-output examples. *ACM Sigplan Notices* 46, 1 (2011), 317–330.

[3] Jan Hajič, Massimiliano Ciaramita, Richard Johansson, Daisuke Kawahara, Maria Antònia Martí, Lluís Màrquez, Adam Meyers, Joakim Nivre, Sebastian Padó, Jan Štepánek, et al. 2009. The CoNLL-2009 shared task: Syntactic and semantic dependencies in multiple languages. (2009).

[4] Arun Iyer, Manohar Jonnalagedda, Suresh Parthasarathy, Arjun Radhakrishna, and Sriram K Rajamani. 2019. Synthesis and machine learning for heterogeneous extraction. In *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation*. 301–315.

[5] Guillaume Lample, Miguel Ballesteros, Sandeep Subramanian, Kazuya Kawakami, and Chris Dyer. 2016. Neural architectures for named entity recognition. *arXiv preprint arXiv:1603.01360* (2016).

[6] Oleksandr Polozov and Sumit Gulwani. 2015. FlashMeta: a framework for inductive program synthesis. In *Proceedings of the 2015 ACM SIGPLAN International Conference on Object-Oriented Programming, Systems, Languages, and Applications*. 107–126.

[7] Stephen Soderland. 1999. Learning information extraction rules for semi-structured and free text. *Machine learning* 34, 1-3 (1999), 233–272.

[8] Emma Strubell, Patrick Verga, David Belanger, and Andrew McCallum. 2017. Fast and accurate entity recognition with iterated dilated convolutions. *arXiv preprint arXiv:1702.02098* (2017).

[9] Paul Viola and Mukund Narasimhan. 2005. Learning to extract information from semi-structured text using a discriminative context free grammar. In *Proceedings of the 28th annual international ACM SIGIR conference on Research and development in information retrieval*. 330–337.

[10] Ruixue Zhang, Wei Yang, Luyun Lin, Zhengkai Tu, Yuqing Xie, Zihang Fu, Yuhao Xie, Luchen Tan, Kun Xiong, and Jimmy Lin. 2020. Rapid Adaptation of BERT for Information Extraction on Domain-Specific Business Documents. *arXiv preprint arXiv:2002.01861* (2020).