Few-Shot Learning for Structured Information Extraction From Form-Like Documents Using a Diff Algorithm

Nerya Or
Google
nerya@google.com

Shlomo Urbach
Google
urbach@google.com

ABSTRACT

We present a novel approach for extracting structured data from a collection of similarly-structured scanned documents (e.g., multiple instances of the same form, or printouts from a database). Documents are not required to have a fixed layout; the position of some elements may shift vertically, and groups of fields can appear repeatedly. We are robust against OCR errors and other noise. Our training stage requires only a handful of sample documents, one of which is annotated for fields of interest. Using this training data, we are able to extract data from other similar documents. Extraction of data is performed using a *diff*-like algorithm over boilerplate text tokens of the documents, which is leveraged to find areas in the input documents which correspond to areas in the annotated document.

KEYWORDS

document understanding, structured data extraction, forms, form understanding, form parsing

1 INTRODUCTION

Collections of similar-looking structured documents are a common occurrence in many fields. Often, a problem of interest is to transform such a collection into a database of structured data (e.g., into a spreadsheet where each document is represented by a single row, and field values in that document are represented as columns).

The scope of this problem is inherently not well-defined, due to the vast number of document types that exist in the world. Nevertheless, we attempt to formalize some of our notations.

Definition 1. *A* template $\mathfrak{T}$ *is a family of documents, all of which are instances of a single type of form, or printouts of records from a single database. Different documents in a template may contain certain variations in their layout, such as vertical shifts and repeating fields (see specific assumptions in Section 3).*

In this work, we present a system capable of solving the following problem:

Problem 1. *We are given a collection of documents from some template $\mathfrak{T}$. Using a single human-annotated document, and a handful of additional non-annotated samples for training, we wish to extract structured data from fields in all of the documents in the collection.*

The small number of training inputs that we require makes our system ideal for use by users that have limited resources, in contrast to ML systems that typically require very large numbers of annotated inputs.

Relevant prior work includes [2, 3, 5–10]. However, these works were less focused on training for specific templates.

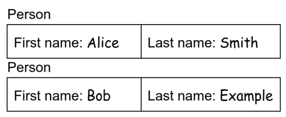
2 DOCUMENT ELEMENTS

Definition 2. *We use the term* boilerplate text *to refer to any text that appears frequently across documents of the template. This includes things like instruction text, section headers, and element labels.*

Documents from the same template share some constant boilerplate text, but usually have different text values filled for various fields.

A document may consist of many elements, and we categorize each as being one of the following:

(1) *Key-value field.* The key is the constant label boilerplate, and the value may change across documents.
(2) *Repeating section.* A section (which may contain key-value fields) which repeats sequentially a variable number of times in each document in $\mathfrak{T}$. We call each repetition of the section an *iteration*. In each iteration, the same boilerplate text appears, including the full text of each key.
(3) *Header* or *footer* text. These are almost-constant chunks of text that may appear at the top or bottom on every page of the document.
(4) Other elements: text which is not associated with any of the above (e.g., legal notices or instructions), tables, and images.


Person

First name: Alice	Last name: Smith

Person

First name: Bob	Last name: Example

Figure 1: A repeating section containing two key-value fields whose keys are "First name" and "Last name". The section has two iterations. Other documents in the same template may contain a different number of iterations.

3 SUPPORTED DOCUMENT LAYOUTS

In the data extraction method presented here, we do not assume that all documents have a fixed layout; some fields may be optional or repeat themselves. As a result, some parts of the documents may not be located in the same place, when inspecting different documents of the same template – elements may float vertically or not exist at all.

In addition, a certain amount of additional text noise (either due to OCR errors or unexpected variations across documents) is tolerated.

We make some basic assumptions about variance among documents in a given template $\mathfrak{T}$:

Assumption 1. *Each key-value field has an area allocated to its value whose size is constant and does not change across documents in $\mathfrak{T}$. In case smaller values appear, they will be surrounded by white-space padding.*

Assumption 2. *Each key-value field has a constant offset vector between the key and the value, across documents in $\mathfrak{T}$.*

Assumption 3. *Horizontal locations of boilerplate text are constant (with slight tolerance for noise) and do not change across documents in $\mathfrak{T}$.*

Assumption 4. *Boilerplate should have a stable order: if documents in $\mathfrak{T}$ contain two boilerplate texts A and B, then whenever both A and B appear in a document, A should come before B (when inspecting the document in a top-to-bottom, left-to-right reading order[1]).*

Assumption 5. *Scanned document pages are not rotated, and not shifted or scaled across the image canvas. Correcting any of these transformations (if needed) can be done as a pre-processing step.*

4 STRUCTURED DATA EXTRACTION

In this section we describe our method for extracting structured data from scanned documents of a certain template. The method is composed of a training stage (Section 4.2), and an extraction stage (Section 4.3).

4.1 Input Document Format

Definition 3. *A* bounding box *is a tuple ⟨top, left, width, height⟩ describing a rectangle on a given document page. Its left and top attributes specify the X and Y coordinates of its top-left corner.*

Definition 4. *A* text token *is a tuple ⟨text, box⟩ containing text of a word, and a bounding box specifying where the text is located.*

Some basic information is assumed to be known about every document we process. In each document $\mathcal{D}$, the text contents are divided into *text tokens*, which we denote as $\mathcal{D}.TextTokens$. Text tokens are stored in a reading order.

4.2 Training

In the first stage of our system, we prepare a trained model which captures important information about a given template $\mathfrak{T}$.

4.2.1 Training Input. The inputs for this stage are a set of n documents $\mathcal{D}_1, \ldots, \mathcal{D}_n \in \mathfrak{T}$. Typically, between $n = 2$ and $n = 10$ documents are enough to yield good results. We can use $n = 1$, if given a clean document containing only boilerplate text.

Our model also includes a *golden document* $\mathcal{G} \in \mathfrak{T}$, which has been annotated to specify fields for extraction.

Definition 5. *A* key-value field annotation $\alpha = \langle key, value, id\rangle$ *describes a key-value field in a given document. It contains two bounding boxes for the key and the value, and a unique identifier.*

Definition 6. *We use* $\mathbf{A}$ *to denote a collection of key-value field annotations within a document.*

[1]This is true for certain languages. For other languages, different reading orders may be applied.

Per Assumption 1, when annotating the box $\alpha.value$ around a value, we require it to be maximal, in the sense that it should cover all the potential area of the value text, across all documents of $\mathfrak{T}$. This should hold even if the actual value text in $\mathcal{G}$ is smaller.

4.2.2 Training Output. As part of our training, we find clusters of estimated boilerplate text tokens which appear frequently across $\mathcal{D}_1, \ldots, \mathcal{D}_n$. This is described in Section 4.2.3. We denote these clusters by $\mathfrak{T}_{clusters}$. We denote our trained model by $\mathbb{M} = \langle \mathfrak{T}_{clusters}, \mathcal{G}, \mathbf{A}\rangle$.

4.2.3 Boilerplate Text Clustering. In this step (outlined in Algorithm 1), we wish to find an approximation of all common boilerplate text tokens in $\mathcal{D}_1, \ldots, \mathcal{D}_n$. By Assumption 3, we note that the two prominent features of a boilerplate token t are $t.text$ and its horizontal location $t.box.left$.

Algorithm 1: FindBoilerplateTextClusters

```
input  : Training set $\mathcal{D}_1, \ldots, \mathcal{D}_n \in \mathfrak{T}$, and a threshold $\gamma$
output : Clusters of text tokens
$tokens \leftarrow$ empty list;
for $i \leftarrow 1$ to $n$ do
    for $t \in \mathcal{D}_i.TextTokens$ do
        $tokens$.Add($t$);
// Cluster by text and X coordinate.
$\mathfrak{T}_{clusters} \leftarrow$ FindClusters($tokens$);
for $c \in \mathfrak{T}_{clusters}$ do
    if $c.size < n\gamma$ then
        remove $c$ from $\mathfrak{T}_{clusters}$;
return $\mathfrak{T}_{clusters}$;
```

To find boilerplate, we run a clustering algorithm *FindClusters* whose inputs are all the text tokens in all of $\mathcal{D}_1, \ldots, \mathcal{D}_n$. *FindClusters* is implemented[2] by (1) partitioning all text tokens into disjoint sets, each set containing tokens with the same *text*, (2) running a *DBScan* algorithm [4] on each set of text tokens, to find clusters by horizontal location of the tokens, and (3) merging clusters of tokens whose *text* is similar, as defined by a heuristic (their lengths are similar, and they have an edit distance lower than a threshold of 25% of the length of the shorter word).

After clusters are obtained from *FindClusters*, any cluster whose size is too small (less than $n\gamma$) is discarded, since it likely represents a token that is not part of the template boilerplate. Specifically, we found $\gamma = 0.9$ provides good results.

We are left with clusters representing text tokens that are likely (but not guaranteed) to be boilerplate. Note that there are cases where we might erroneously classify a token as boilerplate; for example, if a certain value is popular among documents, like the name of a popular city in an "address" field. Nevertheless, our algorithm is able to tolerate a certain amount of noise in boilerplate classification, and so these errors are not critical.

[2]Note that any other clustering method that allows some tolerance over the tokens' text and horizontal location would be suitable.

4.3 Extraction Algorithm

Let $\mathbb{M} = \langle \mathfrak{T}_{clusters}, \mathcal{G}, \mathbf{A} \rangle$ be a model previously trained for some template $\mathfrak{T}$, and let $\mathcal{D} \in \mathfrak{T}$ be a document. In the following sections, we show how one can use $\mathbb{M}$ to extract structured data from $\mathcal{D}$. The output consists of text values of key-value fields, and is keyed by $\alpha.id$ for all field annotations $\alpha \in \mathbf{A}$. Some fields may be missing from the output in case they were not found in $\mathcal{D}$.

We note that this algorithm easily applies also to documents with multiple pages, since we treat each document as one long sequence of tokens. We also note that while the algorithm described here uses one golden input $\langle \mathcal{G}, \mathbf{A} \rangle$, it can be extended to work with multiple golden documents and aggregate their results to remove outliers. Finally, note that the golden document annotations are needed because our boilerplate token classification is only approximate; we need a human in the loop to indicate areas guaranteed to be keys and values.

4.3.1 Extracting Key-Value Fields. Algorithm 2 outlines a procedure for extracting key-value fields in $\mathcal{D}$. Only fields that match those which are annotated in $\mathbf{A}$ are extracted.

Algorithm 2: ExtractStructuredData

```
input  : $\mathbb{M} = \langle \mathfrak{T}_{clusters}, \mathcal{G}, \mathbf{A} \rangle, \mathcal{D}$
output : A mapping in which for each $\alpha \in \mathbf{A}$, $\alpha.id$ is
         associated with its text in $\mathcal{D}$ (if it is found)
$\mathcal{G}_{seq} \leftarrow$ ClassifySequence($\mathcal{G}.TextTokens, \mathfrak{T}_{clusters}$);
$\mathcal{D}_{seq} \leftarrow$ ClassifySequence($\mathcal{D}.TextTokens, \mathfrak{T}_{clusters}$);
$diff \leftarrow$ CalculateDiff($\mathcal{G}_{seq}, \mathcal{D}_{seq}$);
for $\alpha \in \mathbf{A}$ do
    $valueBox_{\mathcal{D}} \leftarrow$ MapValueBox($\alpha, \mathcal{G}, \mathcal{D}, \mathcal{G}_{seq}, \mathcal{D}_{seq}, diff$);
    $value \leftarrow$ GetTextAtBox($valueBox_{\mathcal{D}}, \mathcal{D}$);
    EmitResultValue($\alpha.id, value$);
```

First, for each token in each of the documents $\mathcal{D}$ and $\mathcal{G}$, we apply a cluster classification algorithm, based on the boilerplate text clusters $\mathfrak{T}_{clusters}$. Classification is performed using the token's horizontal location and its text; these are compared against the known clusters in $\mathfrak{T}_{clusters}$. After classification, the token is either classified with a certain *cluster ID*, or not. Tokens associated with a cluster ID are likely to be boilerplate text.

Definition 7. *A* clustered text token *is a 2-tuple containing a text token and a cluster ID to which the token has been classified to.*

We denote by $\mathcal{G}_{seq}, \mathcal{D}_{seq}$ the sequences of clustered text tokens in $\mathcal{G}$ and $\mathcal{D}$, respectively. Each sequence is sorted in reading order, and contains only those text tokens which have been successfully classified to a cluster; text tokens that are not classified to a cluster are omitted from the sequence.

The call to *CalculateDiff()* invokes a longest-common-subsequence algorithm [1] (also known as a *diff* algorithm) over $\mathcal{G}_{seq}$ and $\mathcal{D}_{seq}$. The result is a partial 1:1 mapping from a subset of presumed-boilerplate text tokens in $\mathcal{G}_{seq}$ to presumed-boilerplate text tokens in $\mathcal{D}_{seq}$. Intuitively, such a mapping is a signal that can be leveraged to locate corresponding areas between $\mathcal{G}$ and $\mathcal{D}$.

Finally, we iterate over all key-value annotations $\alpha \in \mathbf{A}$, and for each one, we attempt to find the value of this key-value field in $\mathcal{D}$. We do this by:

(1) Calling *MapValueBox()* in order to estimate the position $valueBox_{\mathcal{D}}$ of the bounding box of the field's value in $\mathcal{D}$ (as will be explained in Section 4.3.2).
(2) Calling *GetTextAtBox()* to extract document text in $\mathcal{D}$ at the estimated location $valueBox_{\mathcal{D}}$. Note that this stage can be applied to non-text values as well, e.g., checkbox elements or images.
(3) Calling *EmitResultValue()* to emit this value in our extraction results, associated with the annotation identifier $\alpha.id$.

Note that *diff* retains the reading-order of the documents, so if, e.g., there are two key-value fields having the same key, we can tell which one is the first and which is the second.

4.3.2 Mapping Positions of Bounding Boxes From $\mathcal{G}$ to $\mathcal{D}$. The *MapValueBox()* procedure in Algorithm 3 is central to our extraction procedure. Given bounding boxes $\alpha.key$, $\alpha.value$ of a key-value field in $\mathcal{G}$, *MapValueBox()* finds the bounding box of the value of the same field in $\mathcal{D}$.

MapValueBox() first estimates the position of the field *key*'s bounding box in $\mathcal{D}$ via a call to *MapKeyBox()*. Next, due to Assumptions 1 and 2, the estimated location of the value box relative to the key is trivially calculated via a call to *ShiftBox()* which applies the X and Y offsets seen in $\mathcal{G}$ between the key and the value.

Algorithm 3: MapValueBox

```
input  : $\alpha \in \mathbf{A}, \mathcal{G}, \mathcal{D}, \mathcal{G}_{seq}, \mathcal{D}_{seq}, diff$
output : Estimated bounding box in $\mathcal{D}$ which corresponds
         to $\alpha.value$
// In case the following line fails, we will
   return early and skip extraction for $\alpha$.
$keyBox_{\mathcal{D}} \leftarrow$ MapKeyBox($\alpha.key, \mathcal{G}, \mathcal{D}, \mathcal{G}_{seq}, \mathcal{D}_{seq}, diff$);
$xOffset \leftarrow \alpha.value.left - \alpha.key.left$;
$yOffset \leftarrow \alpha.value.top - \alpha.key.top$;
return ShiftBox( $keyBox_{\mathcal{D}}, xOffset, yOffset$);
```

MapKeyBox is somewhat elaborate, so we will describe it verbally. It attempts two heuristics, choosing the result from the first one that succeeds, or failing (thus skipping extraction for this key-value field) if both of them fail:

- Observe the set of boilerplate text tokens of $\mathcal{G}_{seq}$ whose bounding boxes are inside $\alpha.key$. If a significant portion of them (e.g,. covering at least 70% of their text characters) are mapped via *diff* to tokens in $\mathcal{D}_{seq}$, then call *Estimate-BoxViaMatches* (Algorithm 4), which uses these mappings to estimate the bounding box $keyBox_{\mathcal{D}}$ of the key in $\mathcal{D}$.
- In case the previous heuristic failed (not enough tokens were matched via *diff*), try a more relaxed approach: look at all boilerplate text tokens of $\mathcal{G}_{seq}$ whose bounding boxes are within a certain vertical distance above and below $\alpha.key$. This range should ideally cover 2-3 text lines. Take the set of all mappings in *diff* that involve those tokens, and attempt to estimate a rough approximation of the bounding

box $keyBox_{\mathcal{D}}$ using them, via another call to Algorithm 4. Next, scale up the approximate $keyBox_{\mathcal{D}}$ box by a factor of ×1.5, to allow some extra slack. Finally, search through all text tokens in $\mathcal{D}$ inside the approximate $keyBox_{\mathcal{D}}$ (including those which were *not* classified as boilerplate). Return successfully only if we found a text match for the original key text from $\alpha.key$ (again, including non-boilerplate classified tokens) inside the approximate $keyBox_{\mathcal{D}}$. We allow minor variations when matching the text, for robustness against OCR errors. The returned $keyBox_{\mathcal{D}}$ will be refined to be the minimal bounding box around the found text tokens.
Note that operating on raw text, as opposed to limiting ourselves only to classified boilerplate text, allows us to overcome possible errors in the boilerplate classification.

Algorithm 4: EstimateBoxViaMatches

```
input : A set of matched bounding boxes in $\mathcal{G}$ and $\mathcal{D}$:
        $\{\langle g_1, d_1\rangle, \ldots, \langle g_m, d_m\rangle\}$, and a bounding box
        $keyBox_{\mathcal{G}}$ in $\mathcal{G}$
output: An estimation of where $keyBox_{\mathcal{G}}$ would be if it
        were in $\mathcal{D}$
xOffsets ← empty list;
yOffsets ← empty list;
for $i \leftarrow 1$ to $m$ do
    xOffsets.Add($d_i.left - g_i.left$);
    yOffsets.Add($d_i.top - g_i.top$);
xOffset ← Median(xOffsets);
yOffset ← Median(yOffsets);
return ShiftBox( $keyBox_{\mathcal{G}}$, xOffset, yOffset);
```

4.4 Supporting Repeating Sections

The algorithm in Section 4.3 extracts simple key-value fields. We now show how it can be extended to extract key-value fields inside *repeating sections*.

First, we add additional data to the annotations **A** in our model[3]: for each repeating section in $\mathcal{G}$, we annotate the area covering a single iteration (denoted the *golden iteration*), and also the area covering all other iterations of the section. These annotated areas must be bounding boxes covering the entire page width. Importantly, we limit annotation of key-value fields in the repeating section to reside only inside the golden iteration.

During extraction, we first delete all tokens in $\mathcal{G}$ that are inside the repeating section, except those that are inside its golden iteration. Then, we need to modify the *CalculateDiff()* algorithm to solve a generalized version of the longest-common-sequence problem, where we allow "star" regex operations over the tokens in the golden iteration. More formally, for a template containing a single repeating section[4], we solve the following problem:

Problem 2. *Let $X = x_1 \ldots x_{|X|}$, $Y = y_1 \ldots y_{|Y|}$ be nonempty strings. Let $1 \le s \le t \le |X|$. We denote $A = x_1 \ldots x_{s-1}$, $B = x_s \ldots x_t$, $C = x_{t+1} \ldots x_{|X|}$ so that X may be written as $X = ABC$ (note that A or C may be empty strings). Observe the family of strings that match the regular expression AB^*C, i.e., $\{AC, ABC, ABBC, ABBBC, \ldots\}$. We are interested in finding some $\chi \in AB^*C$ such that the length of the longest-common-subsequence between χ and Y is maximal (compared to all other $\chi' \in AB^*C$), and to return that subsequence.*

[3]As future work, we can try and detect these annotated vertical areas automatically.
[4]We describe the problem for a single repeating section, but it can easily be generalized to multiple sections.

In our context, $X = \mathcal{G}_{seq}$, $Y = \mathcal{D}_{seq}$, and s, t are determined by golden iteration indices of $\mathcal{G}_{seq}$. While Problem 2 can be solved optimally (e.g., recursively, similar to the classic longest-common-subsequence problem), we opted to implement an approximation: we estimate an upper bound on the number of iterations of the section in $\mathcal{D}$ by counting appearances in $\mathcal{D}_{seq}$ of tokens in the golden iteration, and then duplicate the golden iteration in $\mathcal{G}_{seq}$ as many times. We then use a classic diff algorithm to obtain a one-to-one mapping of the iterations, and we apply heuristics to compact those into a minimal representation: one-to-many, from the golden iteration in $\mathcal{G}$ to all iterations in $\mathcal{D}$. Finally, extracted values that we emit are indexed also by their iteration number.

4.5 Other Document Elements

Headers and footers can be removed as a pre-processing step. Note that this is not mandatory; the diff algorithm is already robust against noise on its own. Similarly, general boilerplate text, images, tables, and other features mentioned in Section 2 are elegantly ignored by the diff algorithm, in most cases.

4.6 Results

We tested the system on about 300 filled forms which were scanned. The template of the forms is fixed and does not contain repeating sections, but some noise exists due to the scanning process. $n = 3$ documents were used for clustering, one of which was annotated for 5 key-value fields. We extracted data from the other documents. Evaluation was performed by comparing the bounding boxes of extracted values vs. human-labeled ground truths. Box comparison was implemented by applying a threshold $\theta = 0.9$ to the area of intersection-over-union of the boxes: $\frac{B_1 \cap B_2}{B_1 \cup B_2}$. Overall precision score was 0.912, and recall was 0.915, yielding an $F1$ score[5] of 0.914. We evaluate bounding boxes as opposed to text contents, since boxes accurately measure the quality of our algorithm's output, while not measuring OCR quality (which is not the focus of this paper).

Sporadic tests were also performed on smaller collections of documents including repeating sections, from templates satisfying the assumptions in Section 3, with comparable results.

REFERENCES

[1] L. Bergroth, H. Hakonen, and T. Raita. 2000. A survey of longest common subsequence algorithms. In *Proceedings Seventh International Symposium on String Processing and Information Retrieval. SPIRE 2000.* 39–48. https://doi.org/10.1109/SPIRE.2000.878178
[2] Brian Davis, Bryan Morse, Scott Cohen, Brian Price, and Chris Tensmeyer. 2019. Deep Visual Template-Free Form Parsing. In *2019 International Conference on Document Analysis and Recognition (ICDAR).* 134–141. https://doi.org/10.1109/ICDAR.2019.00030

[5]True-positives are cases where our predicted box matches the ground-truth with score higher than θ, false-positives are cases where we predicted a value to exist but it failed the match criterion or does not exist in ground-truth, and false-negatives are values that exist in ground-truth but failed the match criterion or were not output at all.

[3] Timo I. Denk and Christian Reisswig. 2019. BERTgrid: Contextualized Embedding for 2D Document Representation and Understanding. *CoRR* abs/1909.04948 (2019). arXiv:1909.04948 http://arxiv.org/abs/1909.04948

[4] Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. 1996. A density-based algorithm for discovering clusters in large spatial databases with noise. AAAI Press, 226–231.

[5] Anoop R Katti, Christian Reisswig, Cordula Guder, Sebastian Brarda, Steffen Bickel, Johannes Höhne, and Jean Baptiste Faddoul. 2018. Chargrid: Towards Understanding 2D Documents. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, Brussels, Belgium, 4459–4469. https://doi.org/10.18653/v1/D18-1476

[6] Xiaojing Liu, Feiyu Gao, Qiong Zhang, and Huasha Zhao. 2019. Graph Convolution for Multimodal Information Extraction from Visually Rich Documents. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Industry Papers)*. Association for Computational Linguistics, Minneapolis, Minnesota, 32–39. https://doi.org/10.18653/v1/N19-2005

[7] Bodhisattwa Prasad Majumder, Navneet Potti, Sandeep Tata, James Bradley Wendt, Qi Zhao, and Marc Najork. 2020. Representation Learning for Information Extraction from Form-like Documents. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*. Association for Computational Linguistics, Online, 6495–6504. https://doi.org/10.18653/v1/2020.acl-main.580

[8] Rasmus Berg Palm, Ole Winther, and Florian Laws. 2017. CloudScan - A Configuration-Free Invoice Analysis System Using Recurrent Neural Networks. In *2017 14th IAPR International Conference on Document Analysis and Recognition (ICDAR)*, Vol. 01. 406–413. https://doi.org/10.1109/ICDAR.2017.74

[9] Daniel Schuster, Klemens Muthmann, Daniel Esser, Alexander Schill, Michael Berger, Christoph Weidling, Kamil Aliyev, and Andreas Hofmeier. 2013. Intellix – End-User Trained Information Extraction for Document Archiving. In *2013 12th International Conference on Document Analysis and Recognition*. 101–105. https://doi.org/10.1109/ICDAR.2013.28

[10] Sandeep Tata, Navneet Potti, James B. Wendt, Lauro Beltrao Costa, Marc Najork, and Beliz Gunel. 2021. Glean: Structured Extractions from Templatic Documents. In *Proceedings of the VLDB Endowment*.