Visual Template Inference for Data Extraction from Documents

YIMING LIN, UC Berkeley, USA
MAWIL HASAN, UC Berkeley, USA
ROHAN KOSALGE, UC Berkeley, USA
ALVIN CHEUNG, UC Berkeley, USA
ADITYA G. PARAMESWARAN, UC Berkeley, USA

Many *templatized documents* are programmatically generated from structured data following a visual template. Such documents include invoices, tax documents, financial reports, and purchase orders. Effective data extraction from these documents is crucial to support downstream analytical tasks. Current data extraction tools often struggle with complex document layouts, incur high latency and/or cost on large datasets, and require significant human effort. The key insight of our tool, TWIX, is to infer the underlying template used to create such documents, and then extract the data, rather than extracting directly from documents. To do so, TWIX first infers the underlying fields, such as columns of tabular portions or keys in co-located key-value pairs, by leveraging their consistent location patterns (e.g., two fields in the same template repeatedly co-occur within a fixed distance apart across multiple records). TWIX then assembles these fields into a template by enforcing visual constraints, such as vertically aligning table rows with their column headers for tabular regions, and horizontally aligning keys with their values for key-value pairs. TWIX then uses this inferred template to accurately and efficiently extract data from templatized documents at a low cost. On one benchmark with 34 diverse real-world datasets, TWIX outperforms state-of-the-art structured data extraction tools (Evaporate, Textract, and Azure Document Intelligence), and vision-based LLMs like GPT-4-Vision, by over 25% in precision and recall. Another benchmark with 30 large datasets demonstrates TWIX's scalability: it is 520× faster and 3,786× cheaper than the most competitive compared tool, for extracting data from large document collections with over 2000 pages.

CCS Concepts: • **Information systems → Document structure**; **Extraction, transformation and loading**.

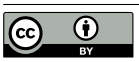
Additional Key Words and Phrases: Document Analytics, Document Data Extraction

ACM Reference Format:
Yiming Lin, Mawil Hasan, Rohan Kosalge, Alvin Cheung, and Aditya G. Parameswaran. 2025. Visual Template Inference for Data Extraction from Documents. *Proc. ACM Manag. Data* 3, 6 (SIGMOD), Article 375 (December 2025), 27 pages. https://doi.org/10.1145/3769840

1 Introduction

Documents (including PDFs or Word files) are ubiquitous across organizations, big and small, and represent some of the most valuable—and yet untapped—sources of insight in data lakes. A particularly common class of documents are those that are *templatized*: documents that are programmatically generated at scale from structured data by employing a *visual template*, i.e., visually rendering each field from a given record in a specific location on the document, and

Authors' Contact Information: Yiming Lin, UC Berkeley, Berkeley, USA, yiminglin@berkeley.edu; Mawil Hasan, UC Berkeley, Berkeley, USA, mawil0721@berkeley.edu; Rohan Kosalge, UC Berkeley, Berkeley, USA, rohankosalge@berkeley.edu; Alvin Cheung, UC Berkeley, Berkeley, USA, akcheung@cs.berkeley.edu; Aditya G. Parameswaran, UC Berkeley, Berkeley, USA, adityagp@berkeley.edu.



ACM 2836-6573/2025/12-ART375
https://doi.org/10.1145/3769840

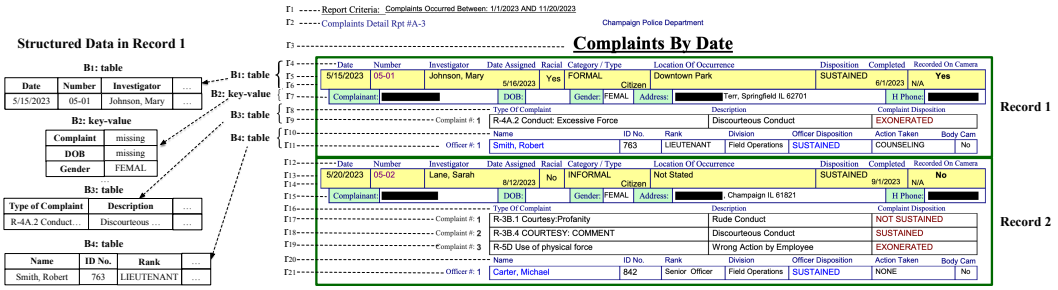


Fig. 1. Police complaint records provided by our collaborators. (Actual values have been replaced for privacy.) Row indices $[r_1, r_2, \dots]$ and block labels $[B_1, B_2, \dots]$ are manually annotated, where B stands for a data block.

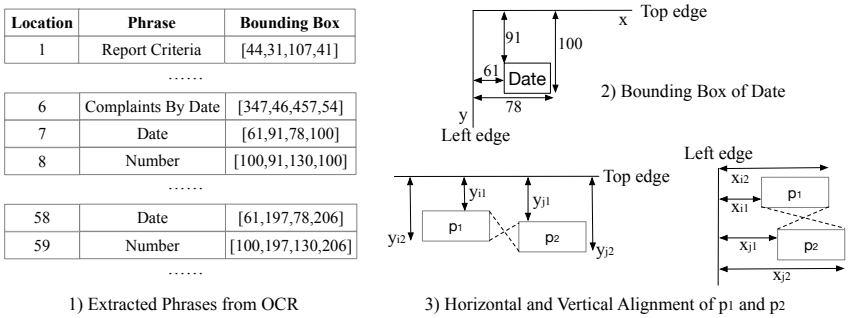


Fig. 2. Extracted Phrases with Locations and Bounding Boxes in Police Complaint Records.

repeating this process for each record—see examples in Figures 1 and 4. This includes tax forms, invoices, financial reports, pay stubs, certification records, expense reports, and purchase orders. Since these documents are programmatically generated, it is easy to generate many pages at scale by simply rendering individual records from one or more relations visually. Indeed, considering just invoices alone, recent estimates forecast over 0.5 trillion invoices generated annually [12]. So, *given a collection of templatized documents, can we cheaply, efficiently, and accurately extract all structured data from it?* Doing this will help us effectively perform Extract-Transform-Load (ETL) from such document collections, with the corresponding structured data outputs stored in data warehouses for subsequent analysis.

Document data extraction is still a pain point. Intuitively, the sheer heterogeneity of templatized documents presents challenges for automated approaches. Templatized documents typically contain a mix of tabular and key-value blocks. Figure 1 shows an example from police complaint records from our collaborators Big Local News at Stanford University; here, record 1 and 2 are visually similar because they were generated using the same template. This example, as in the invoice example in Figure 4, contains table and key-value blocks, with each requiring a distinct approach for data extraction. In a table block, fields are in the first row (e.g., {`Date`, `Number`, ...} in Figure 1), with corresponding values in each column. A key-value block usually places fields and their corresponding values horizontally within a row (e.g., `Gender` and `Female`). Furthermore, different data blocks may be arranged in a nested structure. In Figure 4, multiple small tables (e.g., B_3 and B_4) are nested within a larger table, B_2. Thus, to interpret each document, ***we need to reason about vertical alignment*** *(of column names with corresponding values per record in a table)*, as well as ***horizontal alignment*** *(of values per column associated with the same record for a table, or of keys and their corresponding co-located values in a key-value block).*

Given the heterogeneity, complexity, and need for reasoning about both horizontal and vertical visual alignments, how can we extract structured data from templatized documents? Past work from the 2000s—mainly from the database community—considered HTML-based extraction (e.g., [16, 29, 47]), which relies on HTML tag hierarchies that are absent from documents, and general information extraction (e.g., [15, 22, 33, 51]), which operates on free text without considering visual structure. Recent efforts on document extraction [14, 45, 48, 52, 57, 58] often require users to specify fields and provide labeled examples, leading to high human effort, and additionally lose relationships between extracted values across fields. Other approaches, employed by recent document processing systems [38, 39, 49, 53], involve vision-based LLMs (e.g., GPT-4 Vision [6]) that can leverage visual features, a text-based LLM operating on OCR (Optical Character Recognition) output of a document, or pretrained document extraction APIs, such as AWS Textract [1] or Azure Document Intelligence [2]. However, these approaches, including those that build on them [17], struggle with complex layouts, achieving only 25%–65% precision and recall on a benchmark of 34 real-world datasets. Most approaches process documents page by page, resulting in high latency and cost, e.g., GPT-4 Vision takes 30+ hours and $50+ to extract data from 2000+ pages. We discuss these approaches in detail in Section 8.

TWIX: Inferring templates prior to extraction. We present TWIX[1], a tool for extracting data from templatized documents that first *reconstructs the template from a set of documents generated using the same template*, and then uses the reconstructed template to extract data from the documents. Unlike approaches that apply LLMs or pretrained APIs to each page in a document collection, once the template is inferred, TWIX no longer requires such calls and can extract structured data quickly, accurately, and at no cost. This is because the complex document can be decomposed, based on the template, into data blocks with simple structures (tables or key-value pairs). In Figure 1, every record, like Record 1, can be decomposed into four blocks: a table in block B_1, a list of key-value pairs in B_2, followed by two tables in B_3 and B_4. Extracting data from these simple blocks, once identified, is easier than operating directly on documents. Moreover, extraction using templates preserves data relationships (e.g., which extracted values belong to the same row) without requiring users to specify attributes or provide labeled data.

A natural question that emerges is: are visual templates already available? Unfortunately, they are often absent in real-world settings. Document creators are typically not the ones analyzing them. For example, our journalism collaborators analyze use-of-force records in Figure 1 produced by police departments, while buyers analyze invoices generated by sellers. Specifying templates is also challenging for non-technical users due to the documents' mixed and heterogeneous structure [24]. In Figures 1 and 4, users may struggle to distinguish between tables and key-value pairs or incorrectly flatten nested tables. Even simply specifying fields is difficult: our datasets contain up to 58 fields per record, making manual specification time-consuming and error-prone.

Template inference remains challenging. While inferring a visual template and then using it for extraction intuitively makes sense, template inference presents challenges in two fronts: *field inference*, i.e., inferring which phrases are fields, and *template assembly*, i.e., assembling the fields into a template used to generate records. Fields refer to columns in tables, e.g., `Date` in B_1, or keys in key-value pairs, e.g., `DOB` in B_2. Inferring fields in templatized documents is non-trivial due to complex document layouts. Real-world documents often contain a mix of nested key-value and tabular blocks, interspersed with metadata (e.g., headers, titles). While one could use visual indicators that are easy for a human to understand, e.g., vertical or horizontal alignment, proximity of value phrases to key ones, or indentation, these are hard to automatically leverage and develop

[1]Short for **T**emplatized document **W**rangling for **I**nformation e**X**traction.

Data	Phrase	Vector	
First 5 Records in Police Complaints	Date	[7,58,128,195,251]	M, Δ=1
	Number	[8,59,129,196,252]	
	5/15/2023	[17]	
	05/01	[18]	
	Type of Complaint	[34,85,155,222,278]	M, Δ=1
	Description	[35,86,156,223,279]	
	No	[74,78,126,140,142,144,187,…]	
First 5 Records in Invoices	Line	[16,126,251,381,503]	PM
	Start Date	[17,39,84,127,149,194,252,…]	M, Δ=1
	End Date	[18,40,85,128,150,195,253,…]	

Fig. 3. Location Vectors of Sample Phrases in First 5 Records in Police Complaints and Invoices. M and PM denote Perfect Match and Partial Perfect Match.

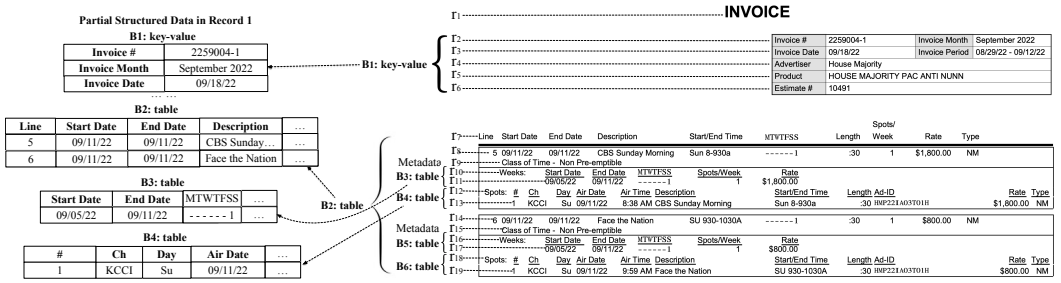


Fig. 4. Portions of Record 1 in an Invoice Document from the Open Benchmark. Row indices $[r_1, r_2, \dots]$ and block labels are manually annotated. There are thousands of invoice records in the document that follow the same template.

rules for. Moreover, inferring fields alone is insufficient to capture the template, which must also encode the structural pattern, i.e., how fields are organized to form records.

Inferring fields based on visual patterns and LLMs. Our key insight is that *fields tend to appear in similar locations across records* (e.g., {`Date`, `Number`, … } in Figure 1), while *the values for the same field can be different across records* (e.g., `05-01` and `05-02` are values of `Number` in Record 1 and 2, respectively). Consider the phrases extracted by a typical OCR tool [5] in Figure 2-1, where the location indicates order of extraction (e.g., `Report Criteria` and `Date` in Record 1 are the 1st and 7th phrases). If we then assemble every location where a given phrase appears into a vector as in Figure 3, for fields like `Date` and `Number`, the difference (or gap Δ) in their location vectors is constant, since `Number` is always the next extracted phrase to `Date` in every record. Among all phrases, typically only a small subset (corresponding to fields) exhibit the consistent location pattern as described above with constant differences (i.e., Δ). Instead, most values (e.g., `05-01`) appear randomly across records and their location vectors are often not consistent. We propose a clustering approach to group phrases with consistent location patterns. When visual cues prove insufficient (e.g., the value for a field is constant across records), we use the semantic knowledge of LLMs to improve robustness. Instead of relying on LLMs to handle complex tasks, such as recognizing intricate visual layouts, we restrict their role to simple questions, like "Is this phrase a field or a value?".

Assembling template based on row labels. To further assemble the template based on inferred fields, our key insight is that a record typically *consists of multiple data blocks, such as table or key-value blocks, and their placement follows a consistent pattern* across records. To infer the template, TWIX assigns each row a label from {*Key, Key-Value, Value, Metadata*}, each associated with a

probability estimated based on the inferred fields. We cast the row labeling problem as one that finds the most probable label per row, constrained by the validity of table structure. For example, a *Key* (e.g., row r_4 in Figure 1) row must have a vertically-aligned *Value* row (e.g., r_5) beneath it, and vice versa. While *Key-Value* rows must have a key for each value, some keys could have missing values. We formalize this *row labeling problem* as an Integer Linear Programming problem (ILP), and prove its **NP-hardness**. We then design efficient pruning strategies on a small number of rows to infer the row labels and then the template.

Once the template is inferred, completing data extraction is straightforward. TWIX divides templatized document into a list of records, and decomposes each record into a list of data blocks with simple structures (i.e., tables or key-value pairs), each of which is easy to extract data from. Overall, we make the following contributions as part of TWIX, a robust tool for data extraction from templatized documents.

- We introduce the concept of a *template* and propose novel algorithms to infer the template for data extraction. Specifically, we develop a clustering-based algorithm that reasons about location patterns, along with LLM input, to infer fields. We further formalize the problem of inferring the template structure as an ILP, show its NP-hardness, and provide an efficient solution.
- We present efficient techniques to extract structured data based on the inferred template at no additional cost.
- We conduct a comprehensive evaluation on two benchmarks comprising 64 diverse real-world datasets, comparing our approach against six state-of-the-art techniques. TWIX achieves around 90% precision and recall, **outperforming each baseline by over 25%** in both metrics. We introduced a new metric called structure precision and recall that accounts for the nested and heterogeneous structure, and once again, TWIX **outperforms the best baseline by over 31% (and 22%) on structure precision (and recall)**. TWIX scales easily to large datasets and is **520× faster** and **3786× cheaper** than the most competitive baseline, on document collections with 2000+ pages.

2 Background and Overview

We focus on a single document D, formed by concatenating all documents $D_1, D_2, \ldots, D_n$ that share the same template. Then D consists of records created using the same template (e.g., Records 1 and 2 in Figure 1) and other metadata (e.g., headers, footers), where the concepts of record, template, and metadata are defined shortly. This setting is common, e.g., invoices, purchase orders, tax documents, financial reports, and immigration forms.

Phrases. We operate on the serialized plain text representation of the concatenated document D. Let P be the phrases extracted from D by using Optical Character Recognition (OCR) tools [5, 8],[2] in ascending order of location, i.e., $P = [p_1, p_2, ..., p_m]$. Figure 2-1 presents phrases extracted from the police complaint document in Figure 1. Each phrase p_i is paired with its bounding box $b_i = [x_{i1}, y_{i1}, x_{i2}, y_{i2}]$ shown in Figure 2-1. Here, x_{i1} and x_{i2} represent distances from the left and right edges of p_i to the left edge of the page, respectively. Similarly, y_{i1} and y_{i2} denote distances from the top and bottom edges of p_i to the top edge of the page, respectively. Figure 2-2 presents the bounding box of the first `Date` (`Date` in Record 1). Each phrase p_i has its *location* i in the document, denoted as $loc(p_i)$, determined by the OCR tool, which extracts phrases row-by-row from left to right, as shown in Figure 2-1. For instance, `Report Criteria` and `Date` (in Record 1 in Figure 1) are the first and seventh phrases extracted with location 1 and 7, respectively. TWIX supports both well-formed and scanned documents. In our Q-Benchmark, scanned (well-formatted resp.)

[2]Our approach is suitable for documents where phrases and their bounding boxes can be extracted, such as PDFs, Word documents, and images of scanned documents.

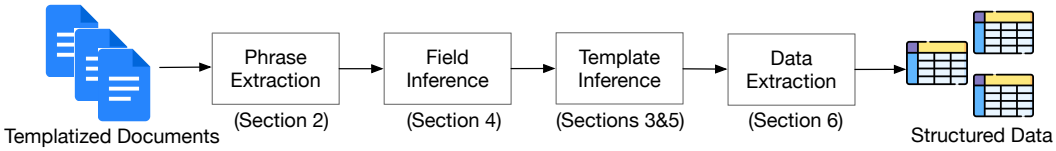


Fig. 5. Overview of the TWIX Pipeline.

documents occupy 22% (78% resp.). TWIX can handle any document as long as the OCR tool can extract phrases and their bounding boxes.

Rows and Blocks. A row is a list of horizontally aligned phrases. Two phrases p_i and p_j (e.g., p_1 and p_2 in Figure 2-3), with bounding boxes $[x_{i1}, y_{i1}, x_{i2}, y_{i2}]$ and $[x_{j1}, y_{j1}, x_{j2}, y_{j2}]$, are visually *horizontally aligned*, if $y_{i1} \leq y_{j2} \wedge y_{i2} \geq y_{j1}$, and are *vertically aligned*, if $x_{j1} \leq x_{i2} \wedge x_{j2} \geq x_{i1}$. We then transform phrases $P = [p_1, p_2, \dots]$ to rows $R = [r_1, r_2, \dots]$ greedily by scanning $p_i \in P$ in increasing order of location. A phrase p_i is merged into an existing row r if p_i is horizontally aligned with every phrase in r; otherwise, a new row is created for p_i. We show all rows in Figure 1. Finally, a block $B = [r_i, r_{i+1}, \dots, r_j]$ is a list of rows in D.

Phrase Labels. We assign each phrase $p \in P$ one of the $\{field, value, metadata\}$ labels. *Fields* refer to the column names of tables or keys in key-value pairs. A *value* is a phrase whose corresponding field can be identified, such as a cell in a table row, or a value in a key-value pair. Note that the same phrase may appear multiple times in P (e.g., `Date` in police complaints appears in row r_4 and r_{12}). Other non-field phrases whose fields cannot be identified are called *metadata*, such as the title "`Complaints By Date`", or headers, e.g., phrases in row r_1 and r_2 in Figure 1.

Row and Block Labels. A row r_i is assigned one of the {*Key*, *Key-Value*, *Value*, *Metadata*} labels. r_i is a *Key* row if it contains no values and each field $p \in r_i$ has at least a vertically aligned value p' beneath p. Similarly, a *Value* row r_i contains no fields, and each value has a vertically aligned field preceding it. r_i is a *Metadata* row if any phrase in r_i is metadata, and in a *Key-Value* row, any value has a preceding field. We now define a *table* or a *key-value* block. A block $B = [r_i, r_{i+1}, \dots, r_j]$ is a table block if r_i is a Key row and other rows in B are all Value rows, where for any value $p \in r_j$ with $j > i$, there exists a field $p' \in r_i$ such that p is vertically aligned with p'. B is a key-value block if any row $r_i \in B$ is a Key-Value row.

Overall TWIX pipeline. Figure 5 shows an overview of TWIX. ① **Phrase Extraction**: Given a concatenated templatized document D, TWIX applies OCR tools to convert D into a list of phrases, each paired with its bounding box and location (e.g., part of the extracted phrases in police complaint records is shown in Figure 2-1). ② **Field Inference**: TWIX clusters the extracted phrases to group phrases that share consistent location patterns into fields, and uses LLMs to provide semantic knowledge and enhance robustness (Section 4). ③ **Template Inference**: We model a template as a tree (Section 3). Given the inferred fields, TWIX infers the row labels as one of {*Key*, *Key-Value*, *Value*, *Metadata*}. We solve the row labeling problem using Integer Linear Programming, with the goal to find the most probable label assignments constrained by visual alignments (Section 5). ④ **Data Extraction**: We use the inferred template to extract data from the entire document D (Section 6). TWIX is implemented in 1200+ lines of Python.

3 Template Formalization

We now formally define templates. A document D consists of a set of *records*, each generated using a *template*. Each record recursively consists of *blocks*. Let T be the *template* used to generate records, where $T = (V, E)$ is an ordered directed tree with an artificial root node. Each non-root node $v \in V - \{root\}$ is associated with a *type* $\in \{Table, Key\text{-}Value\}$, and a set of *fields*. In Figure 6, we present the templates for police complaints and invoices, respectively, where the type and fields for

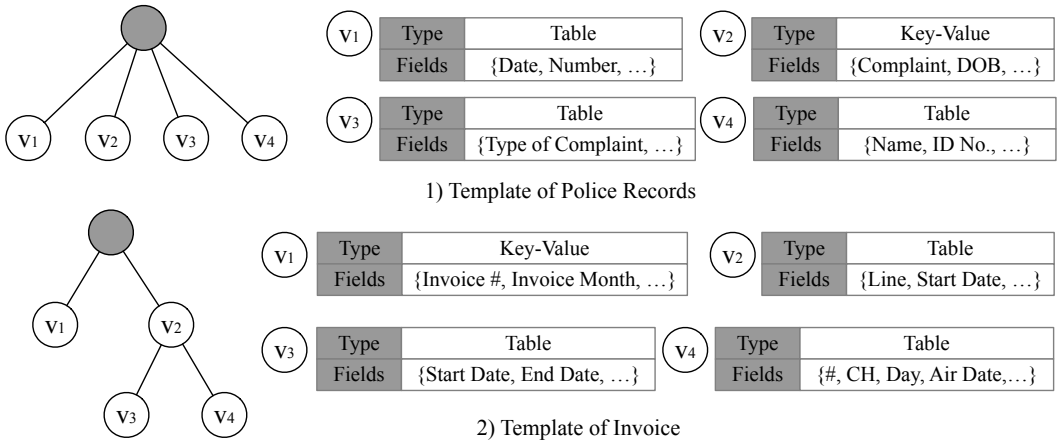


Fig. 6. Templates for Police Records and Invoice Document.

each node are specified. Intuitively, a template defines how a document is populated with records by specifying which fields are populated, in what manner (e.g., as a table or a key-value block), and order (e.g., which block appears first).

For a node $v_i \in V - \{root\}$, we denote $v_i \rightarrow B_i$ to be the process of generating a data block B_i by populating fields in v_i. B_i is a sequence of rows, where every *field* in $v_i.fields$ appears exactly once in B_i and all of the *value* phrases that correspond to those field phrases appear in B_i. Let Rec_i be a record created from template T, comprising a sequence of data blocks generated by nodes in T. To do so, each node v_i in T generates one or more data blocks in the predefined order in T.

Example 1. Block B_1 in police complaint record Rec_1 corresponds to $[r_4, r_5, r_6]$ in Figure 1. In its template in Figure 6-1, B_1 is generated by populating the fields from the table node v_1 in Figure 6-1. A single record then comprises one or more data blocks generated by each of v_1, v_2, v_3 and v_4 in order. In Figure 4, B_3 and B_5 in invoice documents are both generated by v_3 in Figure 6-2, while v_2 generates B_2. Note that v_2 is the parent of v_3, indicating that v_3's data blocks $\{B_3, B_5\}$ are nested within v_2's block B_2, as we will discuss below. □

Let $\mathcal{B}_v^{Rec}$ be the set of blocks generated by the node v in the record Rec, i.e., $\mathcal{B}_v^{Rec} = \{B_i | v \rightarrow B_i, B_i \in Rec\}$. In the police complaint record in Figure 1, $\mathcal{B}_{v_1}^{Rec_1} = \{B_1\}$, while $\mathcal{B}_{v_3}^{Rec_1} = \{B_3, B_5\}$ in the invoice document in Figure 4. Let $loc(\mathcal{B}_v^{Rec})$ be the location of $\mathcal{B}_v^{Rec}$, defined as the smallest (earliest) location of phrase p in any block in $\mathcal{B}_v^{Rec}$, i.e., $loc(\mathcal{B}_v^{Rec}) = \min loc(p), p \in B_i, \forall B_i \in \mathcal{B}_v^{Rec}$.

Two data blocks may overlap (e.g., B_2 and B_3 in Figure 4) if their visual bounding boxes overlap, while others may not (e.g., B_1 and B_2 in Figure 1). To formally define overlapping relationships between data blocks, for two blocks B_i and B_j, we denote $B_i \cap B_j = \emptyset$ if $\forall p_{i_1}, p_{i_2} \in B_i$ and $p_{j_1}, p_{j_2} \in B_j$, such that $(i_1 > j_1) \oplus (i_2 > j_2) = 0$, where i_1, i_2, j_1, j_2 are phrase locations, and $\oplus$ denotes a logical XOR. Otherwise, $B_i \cap B_j \neq \emptyset$. Intuitively, $B_i \cap B_j = \emptyset$ implies that the visual bounding boxes of the two blocks do not overlap, such as any two blocks in police complaint records. In the invoice records in Figure 4, $B_1 \cap B_2 = \emptyset$, while $B_2 \cap B_3 \neq \emptyset$ since phrases from B_2 are placed both before and after B_3, e.g., row r_8 appears before B_3, while row r_{14} appears after B_3. Given a record Rec, and $v_i, v_j \in V$, we denote $\mathcal{B}_{v_i}^{Rec} \cap \mathcal{B}_{v_i}^{Rec} = \emptyset$ if $\forall B_i \in \mathcal{B}_{v_i}^{Rec}, B_j \in \mathcal{B}_{v_j}^{Rec}, B_i \cap B_j = \emptyset$.

Now we describe how edges in the template T influences overlap of data blocks. For any two nodes $v_i, v_j \in V - \{root\}$, when v_i is an *ancestor* of v_j, then $\forall Rec, loc(\mathcal{B}_{v_i}^{Rec}) < loc(\mathcal{B}_{v_j}^{Rec})$ and $\mathcal{B}_{v_i}^{Rec} \cap \mathcal{B}_{v_i}^{Rec} \neq \emptyset$. Likewise, when v_i is a *left sibling* of v_j, then $\forall Rec, loc(\mathcal{B}_{v_i}^{Rec}) < loc(\mathcal{B}_{v_j}^{Rec})$ and $\mathcal{B}_{v_i}^{Rec} \cap \mathcal{B}_{v_i}^{Rec} = \emptyset$.

Example 2. In Figure 6-1, the non-root nodes are placed at the same level from left to right, indicating that their corresponding data blocks B_1, B_2, B_3, and B_4 for the given record do not overlap. In contrast, the template in Figure 6-2 shows that node v_2 is an ancestor of v_3, implying that the data blocks of v_3, $\mathcal{B}_{v_3}^{Rec} = \{B_3, B_5\}$, overlap with the data blocks of v_2, $\{B_2\}$, and B_2 appears before B_3 and B_5. Note that the sequence of data blocks corresponding to the subtree rooted at v_2 can be repeated, provided the sequence of data blocks for its children complies with the edge definition above. □

Given the concepts of templates, data blocks, and records, we next describe how TWIX infers the template (Section 5) by first identifying fields (Section 4), and then extracts data (Section 6).

4 Field Inference

TWIX first infers a set of fields given the extracted phrases P from the document D, as the set of fields often appear in similar locations across records in templatized documents.

4.1 Location Vectors and Matches

Since the same phrase p (e.g., `Date` in Figure 2-1) may appear multiple times in P, such as p_7, p_{58}, we denote $v_p = [i, \ldots, j]$ as the *location vector* of p, comprising the locations of occurrences of p in ascending order. Figure 3 lists the location vectors for sample phrases in the first five records in police complaints and invoices. `Date` appears exactly once in each record, corresponding to the list of phrases $[p_7, \ldots, p_{251}]$, and thus has the location vector $[7, \ldots, 251]$. Let L_p be the length of the vector v_p.

To formalize the intuition that two related fields p_i and p_j share similar locations, their location vectors are often *related by a constant shift*. In Figure 3, adding one to each entry in the location vector for `Date` aligns it with the location vector for `Number`. This is because `Date` and `Number` are fields in the same table node, and thus appear in sync in the blocks corresponding to that node, resulting in constant relative distances across records. We define the concept of *perfect match* to capture this observation.

Definition 1. Perfect Match. Let $v_{p_i} = [i_1, i_2, ...]$ and $v_{p_j} = [j_1, j_2, ...]$ be the location vectors of phrases p_i and p_j. We say p_i is a *perfect match* with p_j, denoted by $M(v_{p_i}, v_{p_j}) = 1$, if $L_{p_i} = L_{p_j}$, $L_{p_j} > 1$ and $\exists\Delta$, s.t., $\forall i_k \in v_{p_i}, j_k \in v_{p_j}, |i_k - j_k| = \Delta$.

When $L_{p_j} = 1$, any two phrases will be a perfect match, and thus we enforce $L_{p_j} > 1$ above. When a field (e.g., `Start Date` in Figure 4) appears in multiple data blocks (B_2, B_3) within a record, its location vector may not be perfectly aligned with that of another field (e.g., `Line` in B_2) that appears in a single data block, even though both fields also share similar location patterns. Thus, we further relax this notion below.

Definition 2. Partial Perfect Match. Let $v_{p_i} = [i_1, i_2, ...]$ and $v_{p_j} = [j_1, j_2, ...]$ be the location vectors of phrases p_i and p_j. We say p_i is a partial perfect match with p_j, denoted by $PM(v_{p_i}, v_{p_j}) = 1$, if there exists a subsequence $v'_{p_i} \subseteq v_{p_i}$, $M(v'_{p_i}, v_{p_j}) = 1$.

Example 3. In Figure 3, any pair of phrases among `Date`, `Number`, `Type of Complaint`, and `Description` is a perfect match, while `5/15/2023`, `05/01` and `No` are not a perfect match with any other phrases. In Figure 4, the same field (e.g., `Start Date`) may appear in multiple blocks (e.g., B_2, B_3, and B_5) within a record, while some phrases, such as `Line`, only appear in one block (e.g., B_2). In this case, `Line` is a partial perfect match with `Start Date` because a subsequence of the location vector for `Start Date` perfectly matches that of `Line`. This subsequence corresponds to the appearances of `Start Date` in B_2 every record. □

4.2 Analysis of Location-based Matches

We have shown previously that similar location patterns between fields can be captured by (partial) perfect matches based on their location vectors. Intuitively, most location vectors are "irregular", while a small amount of them are "regular" (i.e., sharing perfect or partial matches). Informally, regular vectors are fields that tend to co-occur together (i.e., appear in documents near each other), whereas irregular vectors correspond to values or metadata that occur randomly. Now, we establish two properties of the match functions above to show their effectiveness in inferring fields.

Given document D, let $T' = (V', E')$ be the *true* template for D, and let F' be the corresponding set of *fields*.

Proposition 1. Given the true template $T' = (V', E')$, when there exists a unique node $v \in V', v.type = table, p_i, p_j \in v.fields$, if $L_{p_i} = L_{p_j}$, then p_i and p_j are a perfect match; if $L_{p_i} > L_{p_j}$, then p_j and p_j are a partial perfect match.

Proposition 1 (proof in [13]) states that if phrases p_i and p_j are both fields of a unique table node in T' (e.g., `Date` and `Number` in police complaints), and their location vectors have the same length, they must be a perfect match. This is because records are generated using the same T', ensuring that the relative distance between p_i and p_j remains constant across all records. Conversely, when their location vectors have different lengths, such as $L_{p_i} > L_{p_j}$, field p_i (e.g., `Start Date` in invoices) may appear in multiple nodes, while p_j (e.g., `Line`) belongs only to a node v. Here, v_{p_j} and the subsequence of p_i's location vector corresponding to its occurrences in blocks generated by the node v, denoted as v'_{p_i}, must be a perfect match, since v'_{p_i} and v_{p_j} share the same vector length.

Next we establish the corresponding property for fields in a key-value node. Consider a true field p in a node v whose type is $Key\text{-}Value$ in T' (e.g., `DOB` in police complaints). Let $f(p) = True$ if the corresponding value for p is always missing or present in every record generated by T' (e.g., `Gender`). Otherwise, $f(p) = False$.

Proposition 2. Consider the true template $T' = (V', E')$. Given a unique node $v \in V', v.type = Key\text{-}Value, p_i, p_j \in v.fields$, and $f(p_i) = f(p_j) = True$: if $L_{p_i} = L_{p_j}$, then p_i and p_j are a perfect match; if $L_{p_i} > L_{p_j}$, then p_j and p_j are a partial perfect match.

Proposition 2 (proof in [13]) states that if two phrases have a consistent filling pattern ($f(\cdot) = True$) in a unique key-value node, they must be a perfect match. For instance, if `DOB`'s value is always missing while `Gender`'s value is always present in every record, then `DOB` and `Gender` are a perfect match, provided their location vectors have the same length. Conversely, if $f(p_i) = False$, the relative distance between p_i and p_j is not consistent across records. However, even in this case, the phrase p_i with $f(p_i) = False$ ends up being a partial perfect match with phrase p_j (where $f(p_j) = True$) if there exists a sequence of two data blocks in which p_i is either consistently "present" or consistently "missing," a scenario commonly observed in practice.

Notably, TWIX does not rely on bounding boxes (i.e., the physical coordinates of phrases in Figure 2-2) to infer fields, which may be brittle due to OCR noise. Instead, TWIX uses phrase locations based on OCR extraction order to more reliably capture consistent field patterns. In Figure 1, the location difference between `Date` and `Number` is consistently 1 every record, as `Number` is always next to `Date`, regardless of their bounding boxes.

4.3 Field Inference Algorithm

Location-based matches are effective for inferring fields as shown by Proposition 1 and 2. However, when a field's value is constant across records (e.g., when the value of `Disposition` is always `SUSTAINED` in police complaints), this value may share similar location vector patterns with its corresponding field. To address this case, we incorporate LLMs to provide semantic knowledge to help distinguish whether a phrase is a value or a field.

We now outline our field inference algorithm in Algorithm 1, which uses location-based matching as the primary method, while incorporating LLMs to enhance robustness and handle edge cases where location vector patterns alone may be misleading.

Step 1: Phrase Clustering. TWIX first merges any pair of phrases p_i and p_j into one cluster if $M(p_i, p_j) = 1$ (Line 2-9). After clustering, true fields that share consistent location vectors are grouped in the same cluster leading to large clusters, whereas value clusters tend to be smaller due to their inconsistent location patterns. Consider police complaints in Figure 1. This step ensures that the set of true fields appearing in the same table block, such as B_1, B_3, and B_4, are merged into one cluster, based on Proposition 1.

Step 2: Cluster Pruning. TWIX prunes value clusters from the resulting clustering based on the observation that value clusters are typically smaller than field clusters. TWIX incorporates cluster size into a confidence width metric explained shortly: larger clusters (likely fields) have lower width, indicating higher confidence, while smaller clusters (likely values) have higher width. To enhance robustness, TWIX also incorporates semantic knowledge provided by LLMs. Each cluster is assigned a probability of being a field or value cluster based on the semantics of its phrases (e.g., `5/15/2023` is more likely a value than `Date`; easily judged by LLMs). Let $|C|$ denote the size of cluster C (i.e., number of phrases), $Pr(C)$ the probability that C is a field cluster, and $w(C)$ the width of the confidence interval, respectively. Estimating $Pr(C)$ requires semantic knowledge provided by LLMs by using the prompt below, where the phrases in C are used to fill the [phrases] placeholder.

```
LLM Prompt: Given a list of phrases labeled as either "key" or "value", identify and
return the phrases that are more likely to be keys. [Phrases]
```

$Pr(C)$ is estimated as the percentage of fields identified by LLMs over all phrases in C. $w(C)$ is computed conditioned on a confidence level of 95% as $w(C) = 2 \times 1.96 \times \sqrt{\frac{Pr(C)(1-Pr(C))}{|C|}}$.

We first remove all singletons, denoted as SL, from the current set of clusters $\mathcal{C}$ (Line 11), as a phrase that does not match with any others is unlikely to be a field. We consider a new graph $G = (V, E)$ on clusters, where $V = \mathcal{C} \setminus SL$. We say cluster C_i *dominates* C_j if $Pr(C_i) > Pr(C_j)$ and $w(C_i) < w(C_j)$, implying that C_i has a higher probability to be a field cluster with more confidence than C_j. For any pair of clusters (C_i, C_j), we add an edge from C_i to C_j if C_i dominates C_j (Line 12-14). Let $\mathcal{C}_p$ be the maximal node set of G, where for any cluster $C \in \mathcal{C}_p$, there does not exist another cluster $C' \in V$ that dominates C. After the graph is constructed by considering every pair of clusters in V, $\mathcal{C}_p$ is returned (Line 15).

Step 3: Cluster Recovery. In Step 2, true field clusters may be pruned if a true field appears in more than one block in a record, resulting in imperfect matches with other true fields. For instance, for the invoice in Figure 4, the phrase `Start Date` appears in B_2, B_3, and B_5, and is not a perfect match but rather a *partial perfect match* with the other unique true fields in B_2. Using Proposition 1, we see that a sub-sequence of the location vector of `Start Date` (i.e., occurrences of `Start Date` in B_2 in every record) is a perfect match with the location vector of another field (e.g., `Line`) in B_2. Here, we recover the clusters (including singletons) that are *partial perfect matches* with the identified clusters but weren't identified in previous steps (Line 18-20).

5 Template Inference

We now describe how TWIX infers the template T given the concatenated document D populated with records generated by T.

5.1 Row Labeling

Given the set of inferred fields F, TWIX next aims to infer the structure of the template by first labeling rows.

Algorithm 1: `Field Inference`

Input: P

/*__Step 1: Phrase Clustering__*/
$C \leftarrow \emptyset$; $P' \leftarrow set(P)$
for $p_i \in P'$ **do**
  $merge\_flag \leftarrow 0$
  for $C \in C$ **do**
    if $\exists p_j \in C$, *s.t.*, $M(v_{p_i}, v_{p_j}) = 1$ **then**
      $C \leftarrow C \cup p_i$; $merge\_flag \leftarrow 1$
  if $merge\_flag = 0$ **then**
    $C \leftarrow \{p_i\}$, $C \leftarrow C \cup C$
/*__Step 2: Cluster Pruning__*/
$G \leftarrow (V, E)$; $V \leftarrow C \setminus SL$; $E \leftarrow \emptyset$
for $(C_i, C_j), C_i, C_j \in V$ **do**
  if $Pr(C_i) > Pr(C_j)$ *and* $w(C_i) < w(C_j)$ **then**
    $E \leftarrow E \cup (C_i, C_j)$
$C_p \leftarrow Maximal(V)$
/*__Step 3: Cluster Recovery__*/
$F \leftarrow C_p$
for $C_i \in C_p, C_j \in C \setminus C_p$ **do**
  if $\exists p_i \in C_i, p_j \in C_j, L_{p_i} \leq L_{p_j}$, *s.t.*, $PM(v_{p_i}, v_{p_j}) = 1$ **then**
    $F \leftarrow F \cup C_j$
Return F

5.1.1 Row Label Probabilities and Alignment. Consider the list of phrases P extracted from D, and let $R = [r_1, r_2, ...]$ be the set of rows in D where row r_i represents a list of phrases in a row. Each row $r \in R$ is assigned with one of the four labels, $\{\texttt{K}, \texttt{V}, \texttt{KV}, \texttt{M}\}$[3], representing Key, $Value$, $Key\text{-}Value$, and $Metadata$, respectively. In Figure 7, r_4 is a Key row, r_5 and r_6 are $Value$ rows, r_7 is a $Key\text{-}Value$ row, and r_3 is a $Metadata$ row.

Definition 3. Row Label Probabilities. Consider a row $r = [p_1, p_2, ..., p_n]$. Let $Prob^r_K$, $Prob^r_V$, $Prob^r_{KV}$ and $Prob^r_M$ be the probability of the row r having the label K, V, KV and M, respectively.

$$Prob^r_K = \frac{1}{m} \sum_{1 \leq i \leq n-1} I(p_i, p_{i+1}, K) \tag{1}$$

$$Prob^r_V = \frac{1}{m} \sum_{1 \leq i \leq n-1} I(p_i, p_{i+1}, V) \tag{2}$$

$$Prob^r_{KV} = \frac{1}{m} \sum_{1 \leq i \leq n-1} I(p_i, p_{i+1}, KV) \tag{3}$$

where $m = \sum_{1 \leq i \leq n-1} I(p_i, p_{i+1}, K) + I(p_i, p_{i+1}, V) + I(p_i, p_{i+1}, KV)$
Here I is an indicator function, with $I(p_i, p_{i+1}, K) = 1$ if and only if $p_i \in F$ and $p_{i+1} \in F$, denoting that the pair of phrases (p_i, p_{i+1}) are both fields. Similarly, $I(p_i, p_{i+1}, V) = 1$ and $I(p_i, p_{i+1}, KV) = 1$ denoting that the phrase pair (p_i, p_{i+1}) are both values (i.e., $p_i \notin F$ and $p_{i+1} \notin F$) and a key-value pair (i.e., $p_i \in F$ and $p_{i+1} \notin F$). Additionally, we set $Prob^r_M = \epsilon$, a small positive number. ($\epsilon = 0.0001$

[3]While keys and fields are analogous, we use different terminologies to indicate labeling of rows and phrases respectively for clarity.

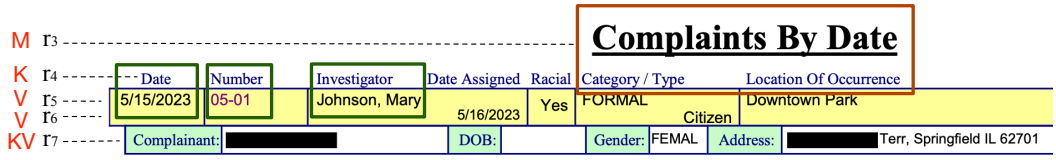


Fig. 7. Row Labels and Vertical Alignments in a Subportion of Police Records.

in our implementation.) We will explain the rationale for this probability shortly. We then normalize the probabilities by dividing each probability $Prob^r_x, x \in \{\texttt{K}, \texttt{V}, \texttt{KV}, \texttt{M}\}$ by $(1+\epsilon)$. Including ϵ makes a negligible impact on the probability distribution across the labels K, V, and KV as ϵ is very small.

Example 4. In Figure 7, consider row r_4 and assume all the phrases in r_4 are inferred as fields. $Prob^{r_4}_K$ is computed by looking at every consecutive pair of phrases in r_4, such as (`Date, Number`), (`Number, Investigator`), (`Investigator, Date Assigned`). Since all phrases are inferred as fields, $I(p_i, p_{i+1}, K) = 1$ for all consecutive phrase pairs, and thus $Prob^{r_4}_K = \frac{1}{1+\epsilon}$, and $Prob^{r_4}_M = \frac{\epsilon}{1+\epsilon}$, with $Prob^{r_4}_V = Prob^{r_4}_{KV} = 0$. Assume `Yes` and `SUSTAINED` in r_5 are false positives (i.e., they are values but are incorrectly inferred as fields). Then $Prob^{r_5}_V = \frac{3}{5\times(1+\epsilon)}$, while $Prob^{r_5}_K = \frac{1}{5\times(1+\epsilon)}$ and $Prob^{r_5}_{KV} = \frac{1}{5\times(1+\epsilon)}$. Note that the total number of pairs considered in the denominator in r_5 is different from r_4 since there are two missing values and two phrase pairs (p_i, p_{i+1}) where $p_i \notin F$ and $p_{i+1} \in F$, not considered in the label pool. □

Intuitively, in any label assignment, **each value row in a true table block should be vertically aligned with its header row**. In Figure 7, rows r_5 and r_6 should be visually aligned with r_4. We now define the row alignment based on bounding boxes of phrases in the row. Recall that we have defined the vertical alignment between two phrases p_i and p_j, with bounding boxes $[x_{i1}, y_{i1}, x_{i2}, y_{i2}]$ and $[x_{j1}, y_{j1}, x_{j2}, y_{j2}]$ in Section 2, where p_i and p_j are vertically aligned, denoted as $p_i \Vdash p_j$, if $x_{j1} \leq x_{i2} \wedge x_{j2} \geq x_{i1}$. For example, in Figure 7, `Date` $\Vdash$ `5/15/2023` since their bounding boxes overlap vertically, while `Date` $\nVdash$ `05-01`. Note that vertical alignment based on bounding boxes is robust to minor OCR inaccuracies, as bounding boxes, if imprecise, are typically only slightly oversized and are still reliable to determine vertical alignment.

Definition 4. Row Alignment. Two rows r_i and r_j are *well-aligned*, i.e., $A(r_i, r_j) = 1$, if $\nexists p_{j_k} \in r_j$, s.t., $p_{j_k} \Vdash p_{i_1}$ and $p_{j_k} \Vdash p_{i_2}, p_{i_1}, p_{i_2} \in r_i, i_1 \neq i_2$, and vice versa. Otherwise, $A(r_i, r_j) = 0$.

$A(r_i, r_j) = 1$ simply implies that there does not exist a phrase in a row, say r_i, that overlaps with more than two phrases in the other row r_j. For a Key row r_i and its Value row r_j in the same table, r_i and r_j are typically well-aligned, as no field is usually vertically aligned with more than one value. In Figure 7, $A(r_4, r_5) = 1$ since there does not exist a phrase in r_5 that is aligned with two phrases in r_4, and vice versa. $A(r_4, r_7) = 0$ since `Complaint` is vertically aligned with both `Date` and `Number`. Similarly, $A(r_3, r_4) = 0$ since `Complaints By Date` is aligned with two phrases in r_4.

5.1.2 *The Row Label Assignment Problem.*

We now formally state our problem of row label assignment.

Definition 5. Row Labeling. Consider rows $R = [r_1, r_2, ...]$, and inferred fields F. We introduce variables y^K_i, y^V_i, y^{KV}_i, and y^M_i for row r_i, where $y^K_i = 1$ implies that r_i has label `K`; otherwise, $y^K_i = 0$. The problem of template structure inference is as follows:

$$\max \prod_{r_i \in R} (y_i^K Prob_K^{r_i} + y_i^V Prob_V^{r_i} + y_i^{KV} Prob_{KV}^{r_i} + y_i^M Prob_M^{r_i}) \tag{4}$$

$$\text{s.t.} : \forall r_i \in R, y_i^K, y_i^V, y_i^{KV}, y_i^M \in \{0, 1\} \tag{5}$$

$$\forall r_i \in R, y_i^K + y_i^V + y_i^{KV} + y_i^M = 1 \tag{6}$$

$$\forall i, y_i^K \leq \sum_{j>i} A(r_i, r_j) y_j^V \tag{7}$$

$$\forall i, y_i^V \leq \sum_{j<i} A(r_j, r_i) y_j^K \tag{8}$$

The row labeling problem in Definition 5 aims to find the *most probable row label assignments* by maximizing the products of the probabilities from all rows in R as in (4), under various constraints. Constraints (5) and (6) ensure that each row is assigned exactly one label. Constraint (7) states that for each *Key* row r_i, there must exist a *Value* row r_j under r_i ($j > i$) aligned with it ($A(r_i, r_j) = 1$). Similarly, Constraint (8) says that for each *Value* row r_j, we expect a *Key* row r_i before r_j ($i < j$) with $A(r_i, r_j) = 1$, i.e., the values are aligned with keys.

If the *Metadata* label is not introduced, a row r_i could be assigned a label with 0 probability if it violates Constraints (7) and (8). For example, if r_i has $Prob_K^{r_i} = 1$ but lacks a corresponding $Value$ row beneath it, it would be assigned a label with 0 probability, leading to a poor assignment. To prevent this, we introduce the *Metadata* label with a low ϵ probability, ensuring the worst possible label for a row is *Metadata*. This adjustment has minimal impact on other rows, as a *Metadata* row does not affect the template structure (e.g., inferring table or key-value blocks), nor does it interfere with the assignment of labels with non-zero probability. Empirical observations confirm the effectiveness of the *Metadata* label. True *Metadata* rows are often misclassified as *Key* rows (e.g., r_3 in police complaints) or *Value* rows (e.g., r_9 in invoices). Such misclassified rows are typically *misaligned* with true *Key* rows, allowing them to be easily identified.

Since the objective in (4) is non-linear, we convert it to be linear below for ease of optimization.

Theorem 1. The following is equivalent to Eq. (4),

$$\max \sum_{r_i \in R} (y_i^K \log(Prob_K^{r_i}) + y_i^V \log(Prob_V^{r_i}) + y_i^{KV} \log(Prob_{KV}^{r_i}) + y_i^M \log(Prob_M^{r_i})) \tag{9}$$

We show the proof of Theorem 1 in the technical report [13].

5.1.3 Solving Row Label Assignment: ILP and Hardness. After linearization, the problem defined in Definition 5 with the new objective in (9) is an Integer Linear Programming (ILP) problem. We additionally can show the following:

Theorem 2. Row labeling is NP-hard.

We prove Theorem 2 via a reduction from Vertex Cover in [13]. Unfortunately, solving the above ILP can be expensive.

Pruning Strategy. Thankfully, inferring the template does not require all of D. Instead, we can use a consecutive subsequence, $r_i, r_{i+1}, ..., r_{i+j}$, of rows R containing *at least one record* from the true template. Let $R' \subseteq R$ be the smallest consecutive subsequence of rows in R such that each inferred field $p \in F$ appears at least twice in R'. R' is obtained by scanning rows of R in ascending order and stopping once each inferred field appears twice. We then have:

Theorem 3. Let F' be the true fields in the true template T', and F the inferred fields. If $\exists p \in F \cap F'$, and $\forall Rec$, p appears exactly once in Rec, then R' contains at least one record Rec created by T'.

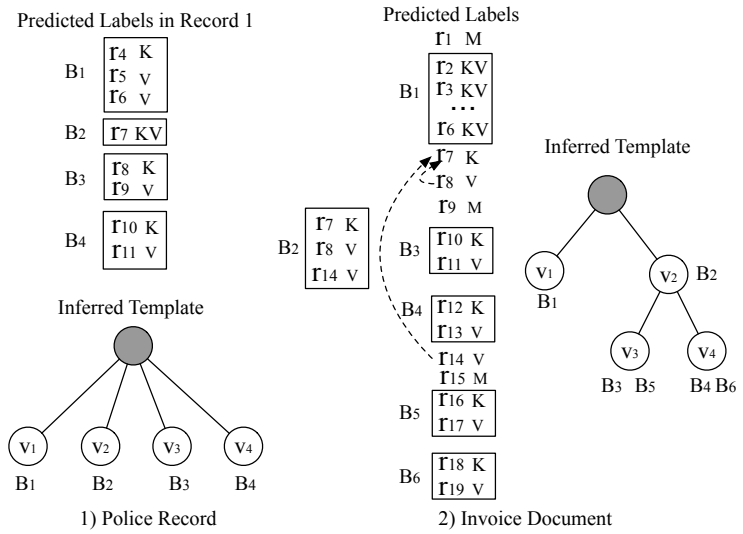


Fig. 8. Template Structure Inference.

Theorem 3 guarantees that as long as there is a field p that is correctly inferred and appears exactly once in every record, then R' contains at least one complete record. See proof in [13].

TWIX considers R' as input to the row labeling problem, and further uses the output row labels to infer the template in Section 5.2. Empirically, thanks to the small size of input rows $|R'|$ (roughly 32 rows on average in our benchmark), an ILP solver [9] provides a solution in milliseconds.

5.2 Template Inference

Given $R' = [r_1, r_2, \ldots, r_n]$ with row label inferences, we aim to determine the structure of the underlying template. Let $B.phrases$ be the set of phrases in a data block B.

We begin by initializing an empty template tree $T = (V, E)$ with an artificial root. 1) For every row r_i labeled K, we create a tree node v_i with $v_i.type = table$ and $v_i.fields = r_i$. 2) We merge together as many consecutive rows labeled KV as possible into a block B, and create the corresponding node v_j where $v_j.type = Key\text{-}Value$ and $v_j.fields = F \cap B.phrases$. 3) For each row r_i labeled as V, we assign it to its *closest aligned key row* preceding r_i, denoted as r_j, where $j < i$, $r_j.label =$ K, $A(r_j, r_i) = 1$, and $\nexists r_k, j < k < i$, such that $r_k.label =$ K and $A(r_k, r_i) = 1$. Note that if a newly created node v_i has the same $v_i.type$ and $v_i.fields$ as an existing node v_j in T, v_i will not be inserted into T. Below, we use two examples to illustrate node creation and tree assembly.

Example 5. In Figure 8, inferred row labels are shown next to the rows. Rows within the same table, such as r_1 and r_2 in Figure 8-1, are aligned and presented together in a block. Rows grouped in the same table block are well-aligned according to Definition 4. Consider police complaints in Figure 8-1. Rows labeled K, specifically r_4, r_8, and r_{10}, trigger creation of nodes v_1, v_3, and v_4, respectively, while r_7, labeled KV, corresponds to v_2. The blocks are sequentially placed with no overlap, with all nodes as leaves. The pre-order traversal of the corresponding nodes in the tree is sorted by block indices (e.g., $loc(B_1) < loc(B_2)$), resulting in the template T depicted in Figure 8-1. In the invoice in Figure 8-2, consecutive KV rows (r_2 to r_6) are merged into a single KV block B_1. In table block B_2, r_7 is the closest key row aligned with r_{14}, making r_{14} a value row in B_2. B_3 and B_5 correspond to the same node v_3 since $r_{17} = v_3.fields$. Therefore, v_3 is created only once, triggered by r_{10}. Since data block B_2 (corresponding to v_2) overlaps with B_3 and B_4 (corresponding to v_3 and

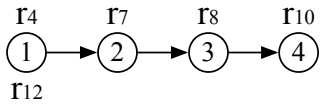


Fig. 9. Record Separation in Police Complaints.

v_4), v_2 becomes the parent of v_3 and v_4, while v_1 is a leaf node since its block doesn't overlap with others. □

The complexity of template inference based on inferred row labels is $O(|R'|^2)$, which is efficient since $|R'|$ is typically small.

6 Data Extraction

Given the inferred template T and the concatenated document D, TWIX aims to extract data from the set of records $\mathcal{R}ec = [Rec_1, Rec_2, \dots]$ generated by T within D. To do so, we first separate the given document D to into records based on T (Section 6.1). Within each record, we further identify the data blocks (Section 6.2), and then extract data from table and key-value blocks (Section 6.3).

6.1 Record Separation

Consider the row representation R of D, $R = [r_1, r_2, \dots]$. Recall that a block B is a sequence of rows. For any node $v \in V$ in the template T, we say that node v is *visited* by block B if all the fields of v appear in block B, i.e., $v.fields \subseteq B.phrases$, denoted by $vis(v, B) = \text{True}$. Intuitively, a record is the smallest block that visits *every* node in T *at least once* via a *pre-order* traversal. The document D is now separated into a list of records $\mathcal{R}ec$.

Example 6. Given the inferred template T in Figure 8-1, we aim to separate records 1 and 2 in Figure 1. We scan each row $r_i \in D$ in ascending order of location, and consider the pre-order traversal of nodes in T in Figure 9. $vis(v_1, [r_4])$ is True as $v_1.fields = r_4$ (r_4 is the header of the table block corresponding to v_1), and v_1 is thus visited. Similarly, $[r_7]$ visits v_2 as $v_2.fields \subseteq r_7$, and $[r_{10}]$ visits v_4. When r_{12} visits v_1 the second time ($v_1.fields = r_{12}$), all nodes in T have been visited at least once, and v_1 being visited by r_{12} the second time implies the start of a new record. Note that it is not necessary that each node is visited exactly once, because nested nodes (e.g., v_3 in invoice template) may be visited multiple times. □

6.2 Block Separation

Given a record $Rec \in \mathcal{R}ec$ from the previous step, we aim to identify a list of blocks corresponding to each node in T. Figures 1 and 4 show the data blocks within a record in police complaints and invoices, respectively.

Block separation proceeds in two steps as in Algorithm 2. First, we assign the labels for each row $r \in Rec$ based on the template T. Given a row $r = [p_i, \dots, p_j]$, if $\exists v \in V, v.type = \textit{Table}$, $\forall p \in v.fields, p \in r$, then $r.label = K$ (Lines 2-4). Otherwise, if $\exists v \in V, v.type = \textit{Key-Value}$, $r \cap v.field \neq \emptyset$, $r.label = KV$ (Lines 5-6). In all the other cases, $r.label = V$ (Lines 7-8). Second, for a value row r_i whose label is V, let r_j be the closest key row preceding r_i. If r_i is aligned with r_j, i.e., $A(r_j, r_i) = 1$, then r_i is assigned to be a value row of the table with the key row as r_j (Lines 11-14). Otherwise, if there exists another key row r_l preceding and as close as possible to r_i, where $A(r_l, r_i) = 1$ and the corresponding node v is not a leaf, then r_i is assigned to be a value row of r_l (Lines 15-18). In other cases, r_i is labeled as *Metadata* (Lines 19-20), as it is neither aligned with any K row (and thus not a valid V row) nor inferred to be a KV or K row. Finally, for any consecutive key-value rows, we merge them together into one block (Lines 21-23).

Example 7. Consider the police complaints in Figure 1 and its inferred template in Figure 8-1. The labels for r_4, r_8, and r_{10} are K, r_7's label is KV, and all other rows are labeled V. r_5 is assigned

Algorithm 2: `Block_Separation`$(Rec, T = (V, E))$

```
/*Step 1: Row Label Assignment*/
for r ∈ Rec do
    if ∃v ∈ V, v.type =table, v.fields = r then
        r.label = K
    else if ∃v ∈ V, v.type =key-value, r ∩ v.field ≠ ∅ then
        r.label = KV
    else
        r.label =V
/*Step 2: Block Separation*/
Member ← ∅
for r_i ∈ Rec, r_i.label =V do
    r_j ← closest_preceding_key_row(r_i)
    if A(r_i, r_j) = 1 then
        Member(r_j).append(r_i)
    else
        r_l ← closest_preceding_aligned_key_row(r_i)
        if ∃r_l, v_l.hasChild = True then
            Member(r_l).append(r_i)
        else
            r_i.label = Metadata
for r_i, r_{i+1} ∈ Rec do
    if r_i.label =KV and r_{i+1}.label =KV then
        Member(r_i).append(r_{i+1})
Return Member
```

to r_4 because r_4 is aligned with r_5 and is the closest preceding key row. Similarly, r_9 is assigned to r_8. Now consider the invoice document in Figure 4 and its inferred template in Figure 8-2. The closest preceding key row to r_{14} is r_{12}. However, r_{12} is not the key row for r_{14} because they are not aligned. The closest preceding key row aligned with r_{14} is r_7, corresponding to node v_2 in the template. Since v_2 is not a leaf node, its data block can overlap with others as defined in Section 3. Thus, r_{14} is correctly assigned to r_7. □

Each record $Rec \in \mathcal{R}ec$ is now transformed into a list of data blocks, either table or key-value blocks.

6.3 Data Extraction

We then extract data from the table and key-value blocks.

Data Extraction in Table Blocks. Given a table block B, we next extract its contents. Let $B.fields = [f_1, f_2, \ldots, f_n]$ be the list of inferred fields in B sorted in ascending order of location.

The first row r_l in B corresponds to $B.fields$, and each value row $r_i \in B, r_i \neq r_l$ corresponds to a tuple of this table, where $r_i = [p_{i1}, p_{i2}, \ldots, p_{im}]$. Intuitively, values corresponding to a field are typically aligned vertically as a column in a table block. Thus, given a value row $r_i \in B$, if p_{ij} is vertically aligned with f_j, i.e., $p_{ij} \vDash f_j$, then p_{ij} is determined to be a value of f_j. Given a field f_j, if there does not exist a value $p_{ij} \in r$, such that $p_{ij} \vDash f_j$, then f_j's value is *missing* (or *NULL*) for row r_i. Consider data extraction of block B_1 in Record 1 for police complaints in Figure 1. The value for

Algorithm 3: KV-Extract(B)

```
1  seen ← {}; i ← 0; KV ← ∅
2  for p_i ∈ B do
3      seen[i] ← False
4  while i < |B| − 1 do
5      if seen[i] == False then
6          if p_i ∈ B.fields ∧ p_{i+1} ∉ B.fields then
7              KV ← KV ∪ (p_i, p_{i+1})
8              seen[i + 1] ← True
9          if p_i ∈ B.fields ∧ p_{i+1} ∈ B.fields then
10             KV ← KV ∪ (p_i, missing)
11     i ← i + 1
12 Return KV
```

`Completed` is missing in r_5, `05-01` and `Yes` are the values for `Number` and `Recorded On Camera` in r_5, respectively.

Data Extraction in Key-Value Blocks. In a key-value block B, a field either has a corresponding value or may be missing. In Algorithm 3, to extract key-value pairs given inferred fields, we sequentially scan each consecutive phrase pair (p_i, p_{i+1}) in B. We use an array $seen[i]$ to track whether the phrase p_i has been scanned. If (p_i, p_{i+1}) forms a key-value pair, i.e., $p_i \in B.fields \wedge p_{i+1} \notin B.fields$, it is added to the extraction result (Lines 6-7). We then examine the next phrase pair starting from p_{i+2} by setting $seen[i+1]$ to `True`. If both phrases are fields, the phrase p_i is assigned a *missing* value, and we proceed by examining the phrase pair starting with p_{i+1} (Lines 9-10). For example, if `Complaint` and `DOB` in row r_7 in Figure 1 are identified as fields, their values are inferred as `missing`.

Finally, we assemble the extracted data from each block in every record into a tree, where we describe the representation of the data extraction results in [13].

Metadata Preservation. In addition to extracting structured data, TWIX stores metadata and preserves its associations with the extracted data. To achieve this, TWIX stores the bounding box and page number for every metadata phrase. In police complaints in Figure 1, the phrase `Complaint #1` in row r_9 is extracted as metadata since it is neither a table header nor a value (i.e., it lacks an associated column header). Here, TWIX extracts the table with schema {`Type of Complaint, Description, Complaint Disposition`} while preserving spatial relationships between metadata and table content using bounding boxes. For example, the preserved bounding boxes suggest that `Complaint #1` is horizontally aligned with the first row of the extracted table. Users can leverage this alignment to expand the extracted table by adding a new column called `Complaint` and populating the first row with value `1` for it. Since the interpretation and transformation of metadata depends on use cases, TWIX leaves this task to end users. However, TWIX preserves the spatial relationships between metadata and extracted data to enable such transformations.

Correctness of End-to-end Approach. Finally, we establish the correctness of the end-to-end TWIX pipeline in one commonly observed type of documents, that we call *compliant documents*. We leave the definition of compliant documents and the correctness of TWIX in [13].

7 Experimental Evaluation

We evaluate the performance of TWIX on two benchmarks comprising 64 real-world templatized document collections.

Q-Benchmark			
	# of Datasets	**Avg Dataset Size (tokens)**	**Avg # of Doc**
Easy	6	5902	11.2
Medium	21	7813	6.8
Hard	7	6417	5.1
S-Benchmark			
Easy	23	2, 106, 218	2117
Medium	6	2, 070, 688	2191
Hard	1	1, 059, 775	1577

Table 1. Characteristics of 64 Datasets.

7.1 Evaluation Setup

Datasets. We constructed two benchmarks for our evaluation. The first benchmark, *Q-Benchmark*, focuses on evaluating the quality of extraction results. The second benchmark, *S-Benchmark*, examines the scalability of the compared tools. Q-Benchmark consists of 34 real-world datasets from our collaborators, and three open benchmarks [43, 54, 59], spanning diverse domains such as police use of force documents, invoices, grant reports, order bills, certification records, contracts, and trade forms. We randomly sampled 5 to 30 documents per dataset (around 7.4 documents and 7189 tokens on average), based on dataset size, with more samples for smaller datasets and fewer for larger ones to ensure representative coverage. The ground truth for all datasets was manually collected and verified by three human labelers over a month, highlighting the task's complexity even for humans. The S-Benchmark consists of 30 datasets collected from the open benchmark [43], containing around 2,133 documents and over two million tokens per dataset on average. Given its scale, we do not have ground truth for S-Benchmark, so we primarily focus on evaluating latency and cost.

We classify datasets per benchmark into three types based on complexity: `Easy`, `Medium`, and `Hard`. `Easy` datasets have templates with only one node besides the virtual root, meaning each document contains either a pure table or key-value block. For simplicity, we omit the virtual root when discussing nodes in templates. `Medium` datasets feature templates with more than two nodes, all in the same layer (i.e., all nodes are leaves), indicating sequentially placed, non-overlapping data blocks within a record. `Hard` datasets have nodes with children, implying overlapping data blocks, as in invoices in Figure 4. Table 1 summarizes the characteristics of the datasets.

Tools Compared. We compare TWIX with six baselines. This includes Amazon *Textract* [1], and Azure AI Document Intelligence (*AzureDI* for short) [2], which detect and extract tables and key-value pairs from PDFs. We also include two state-of-the-art vision-based LLMs, *vLLM-S* and *vLLM-C*, both using `gpt4vision` [6] from OpenAI. *vLLM-S* uses a prompt to identify the template structure and then extracts data, while *vLLM-C* directly extracts all key-value pairs from each page. For a table, *vLLM-C* extracts each cell and its header as key-value pairs, whereas for key-value blocks, it outputs a list of key-value pairs. We also considered LLMs based on OCR text, where we provide hints that records are generated from the same template and append multiple example records. As vision-based LLMs outperform any text-based LLMs we tried, we omit them below. The prompts for LLMs baselines are in [13]. Finally, we compare TWIX with *Evaporate* [17], a tool that extracts structured data from documents using LLMs. Evaporate operates on OCR (text) output from documents. Evaporate has two versions: *Evaporate-Direct* (*Eva-D* for short), which uses LLMs to directly identify fields and extract values, and *Evaporate-Code* (*Eva-C* for short), which uses LLMs to generate extraction scripts and then applies those scripts to extract values. TWIX uses Pdfplumber [5] as the OCR tool, which can be replaced by alternatives such as Tesseract [10] or MistralOCR [11], which extract phrases along with bounding boxes.

Metrics. We report precision and recall to evaluate the quality of the extracted results. For TWIX, if an extracted object is a table, we convert it to a list of key-value pairs for each cell with its corresponding key in the header. If a cell contains a missing value, we include $(key, missing)$ as

	Precision							Recall						
	Textract	vLLM-S	vLLM-C	AzureDI	Eva-D	Eva-C	TWIX	Textract	vLLM-S	vLLM-C	AzureDI	Eva-D	Eva-C	TWIX
Easy	0.9	0.74	0.73	0.54	0.47	0.32	**0.96**	0.88	0.62	0.68	0.68	0.52	0.25	**0.98**
Medium	0.5	0.63	0.6	0.49	0.41	0.33	**0.89**	0.51	0.57	0.5	0.62	0.33	0.26	**0.9**
Hard	0.38	0.59	0.64	0.35	0.45	0.35	**0.85**	0.44	0.49	0.57	0.66	0.44	0.26	**0.91**

Table 2. Precision and Recall on Easy, Medium and Hard Datasets on Q-Benchmark.

	S-Precision							S-Recall						
	Textract	vLLM-S	vLLM-C	AzureDI	Eva-D	Eva-C	TWIX	Textract	vLLM-S	vLLM-C	AzureDI	Eva-D	Eva-C	TWIX
Easy	0.74	0.68	N/A	0.35	0.27	N/A	**0.94**	0.66	0.61	N/A	0.38	0.32	N/A	**0.92**
Medium	0.61	0.54	N/A	0.42	0.19	N/A	**0.81**	0.64	0.49	N/A	0.55	0.2	N/A	**0.76**
Hard	0.49	0.51	N/A	0.19	0.18	N/A	**0.8**	0.36	0.48	N/A	0.17	0.19	N/A	**0.76**

Table 3. S-Precision and S-Recall on Easy, Medium and Hard Datasets on Q-Benchmark. N/A indicates that the tool is unable to produce output while preserving structures such as tables and key-value pairs.

part of the output. Key-value blocks naturally represent key-value pairs. We similarly transform the extracted results from the baselines into a list of key-value pairs as the extraction result for each document per dataset. For a document D, let KV_p and KV_t be the inferred key-value pairs and true key-value pairs for D. Precision $P_D = \frac{|KV_p \cap KV_t|}{|KV_p|}$, while recall $R_D = \frac{|KV_p \cap KV_t|}{|KV_t|}$. For a dataset $\mathcal{D} = \{D_1, D_2, \dots\}$, $P_{\mathcal{D}}$ and $R_{\mathcal{D}}$ represent the average precision and recall over documents in $\mathcal{D}$. We report average precision and recall on Easy, Medium, and Hard datasets.

We further evaluate how tools predict the *structure* of extracted data using *structure-precision* (*S-Precision*) and *structure-recall* (*S-Recall*). We first define S-Precision and S-Recall, denoted by SP and SR, between a true table T and a predicted table T', which may have different schemas, and then extend it to more complex settings. We present an example in [13] to illustrate SP/SR.

Let the precision and recall between two tuples $t_i \in T$ and $t'_j \in T'$ be P_{ij} and R_{ij}, respectively. A tuple can be viewed as a list of key-value pairs (column-cell pairs), and let KV_i and KV_j be key-value pairs in t_i and t'_j. Then, $P_{ij} = \frac{|KV_i \cap KV_j|}{|KV_i|}$, and $R_{ij} = \frac{|KV_i \cap KV_j|}{|KV_j|}$. To move from individual tuple pairs to tables, we identify a matching M of matched tuple pairs between T and T'. M is computed by finding a minimum-distance matching in the bipartite graph where T and T' each represent a partition. Each tuple is a node and the distance of edge (t_i, t'_j) is $1 - \frac{|KV_i \cap KV_j|}{|KV_i \cup KV_j|}$. We present a dynamic programming algorithm to find M in [13]. Finally, $SP = \sum_{(t_i, t'_j) \in M} P_{ij} / |T'|$ and $SR = \sum_{(t_i, t'_j) \in M} P_{ij} / |T|$. For two key-value blocks, their SP and SR can be computed using the same metric, as a key-value block can be viewed as a table with a single row.

We now extend SP and SR to handle nested and mixed data blocks. Intuitively, evaluating nested tables requires capturing the co-occurrence of a nested table and the tuple it is nested under. For any nested table T, we identify the tuple t it is nested under. We then construct a flattened table from T with a schema formed by concatenating the schemas of t and T, where each tuple is the concatenation of t and a tuple from T. For a tuple without a nested table, we preserve it by appending NULL values under the unmatched columns (i.e., a left outer-join). We recursively flatten tables when multiple levels of nesting exist. For a document D, we concatenate true and predicted data blocks into two wide tables T and T' for ease of evaluation. Concatenating any two tables T_i and T_j is equivalent to performing a full outer join, i.e., combining their schemas and inserting NULLs for unmatched columns. Finally, we report SP and SR over the true and predicated concatenated tables (T, T').

Both P/R and SP/SR are useful: SP/SR emphasizes structure and can heavily penalize upstream errors, while P/R treats all errors equally but does not consider structure. For example, a mistake in a single key-value pair is counted once for P/R but can be penalized as many times as the number of tuples nested below it for SP/SR.

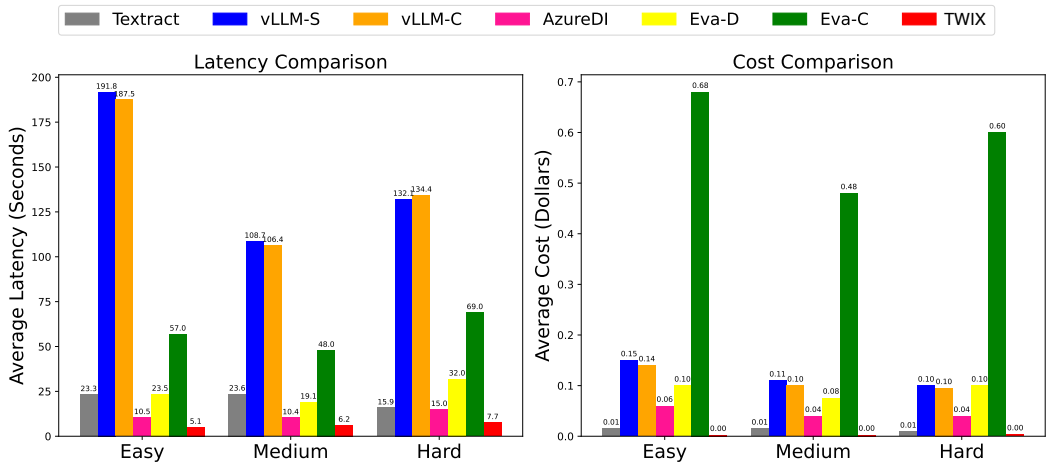


Fig. 10. Latency (Left) and Cost (Right) on Q-Benchmark.

7.2 Experimental Results

Experiment 1: Quality Comparisons. We report the precision and recall for all tools in Table 2 on Q-Benchmark. In `Easy` datasets, both Textract and TWIX perform well, with TWIX outperforming Textract by around 8% in precision and 9% in recall. Vision-based LLM approaches, Evaporate, and AzureDI struggle even on simple templates. Eva-C uses LLM-generated scripts, encoding heuristic rules, to extract field values, which turns out to be not robust. Vision-based LLMs outperform Eva-D by using images instead of text, enabling better use of visual information. We will present a detailed error analysis for all baselines subsequently in Table 5. TWIX significantly outperforms all baselines in both `Medium` and `Hard` datasets, achieving around ***38% higher precision and 39% higher recall compared to Textract, 44% higher precision and 25% higher recall compared to AzureDI, 39% higher precision and 53% higher recall compared to the best version of Evaporate, and 25% higher precision and 33% higher recall compared to the best vision-LLM approach***. We also report structure-precision and structure-recall in Table 2, and TWIX consistently outperforms all baselines on datasets with varying difficulty. Notably, TWIX achieves 31% higher S-Precision and 22% higher S-Recall than the best baseline. TWIX is able to infer complex nested structures (on `Hard` datasets), achieving 80% S-Precision and 76% S-Recall. All baselines struggle with complex layouts, leading to a significant drop in quality. This highlights the importance of inferring the template, making downstream extraction more accurate, compared to general-purpose data extraction solutions that ignore the template.

Experiment 2: Latency Comparisons. We compare the end-to-end latency in Q-Benchmark in Figure 10 (left). Vision LLM approaches are time-consuming, as they process each page as an image. Their latency depends on the number of pages and the size of the output tokens. Data extraction tasks typically return a large number of output tokens, making image-based extraction costly. Textract and AzureDI are much faster than vLLM-based baselines, taking around 21 and 11s to extract data for documents with an average of 10.4 pages per dataset, respectively. TWIX is the most efficient tool, taking around 5s to process a dataset—***2× faster than AzureDI, 4× faster than Textract, 7× faster than Evaporate, and 28.6× faster than vLLM-based approaches***.

Experiment 3: Cost Comparisons. Figure 10 presents the end-to-end cost of all tools in the Q-Benchmark (right). Most baselines charge per page. Textract and AzureDI have fixed rates of $1.50 and $10 per 1000 pages, respectively [1, 2], while GPT-4-Vision APIs charge based on the number of pages and image resolution (treating each page as an image) [7] and Eva-D using GPT-4o charges

	Latency (Minutes)				Cost (Dollars)			
Tools	Easy	Medium	Hard	AVG	Easy	Medium	Hard	AVG
Textract	59	61	41	59	31.8	32.9	23.7	31.6
vLLM-S	1977	1810	876	1901	54.5	52.3	43	54
vLLM-C	2195	2119	1008	2133	53.2	50.8	41.9	52.4
AzureDI	82	49	30	74	21.1	21.9	15.8	21.1
Eva-D	48	54	47	48	24.5	27.1	13.6	24.1
Eva-C	9	11	10	9	4.3	4.2	3.9	4.3
TWIX	**4.2**	**4.3**	**3.3**	**4.1**	**0.014**	**0.015**	**0.012**	**0.014**

Table 4. Latency and Cost on S-Benchmark.

based on the number of tokens in the datasets. Eva-C uses LLMs to generate scripts for extracting values for each field, taking raw text converted from documents as input. While data extraction using the scripts is free, the script generation step incurs high cost. TWIX incurs costs only during template inference, where LLMs filter non-field phrase clusters. The subsequent template-based extraction is LLM-free, incurring no further cost. Across all datasets, TWIX achieves an average cost of less than $0.003 per dataset, representing ***23% of Textract's cost, 6.4% of AzureDI's cost, 1.2% of Evaporate's cost*, and *0.8% of vision-LLM-based tools' cost***.

Experiment 4: Scalability Comparisons. To evaluate how the tools scale to large datasets, we evaluate their cost and latency on S-Benchmark, around 300× larger than Q-Benchmark, in Table 4.

TWIX can scale to large datasets easily with around 4 minutes and $0.014 per dataset. This is because TWIX infers the template from a small portion of the document determined by our pruning strategy and then extracts data for remaining documents based on the inferred template without invoking LLMs. TWIX incurs no additional cost as the number of documents grows and introduces negligible latency since template-based data extraction is inexpensive. Eva-C uses LLMs to generate scripts that extract values for fields. Similar to TWIX, their data extraction is no-cost using scripts. Even in this case, TWIX still achieves noticeable advantages in both latency and cost, while also delivering over 50% higher precision and recall based on the results in Q-Benchmark. All other baselines' latency and cost increase linearly with the number of documents. Vision-based LLMs, for example, take over 30 hours to process around 2000+ pages at over $50. ***TWIX completes the task in a few minutes—ranging from 12× to 174× faster and 1500× to 2423× cheaper than all baselines except Eva-C.***

Experiment 5: Time and cost breakdown of TWIX. TWIX consists of three components: 1) phrase extraction extracting text from documents using OCR tools; 2) template inference, where TWIX infers fields and structure of the template; and 3) data extraction using the inferred template. We provide a breakdown of latency for these three components in Figure 11 on both benchmarks. The numbers represent the percentage of latency for each component.

When the dataset is relatively small in Q-Benchmark, template inference consumes around 85% of the time in Easy, Medium, and Hard datasets, and data extraction is the fastest component. However, as the dataset size grows, as in S-Benchmark, the most time-consuming component becomes phrase extraction, as its time increases linearly with the number of documents, while the latency of template inference remains constant since TWIX infers the template from a small document portion, thanks to the effective pruning strategy. In terms of cost breakdown, all costs in TWIX are incurred during the template inference stage where LLMs are used.

Experiment 6: Performance of Field Inference. We examine the performance of the field inference in TWIX for Q-Benchmark, and report precision and recall in Figure 12.

In Easy datasets with simple template structure, the precision and recall of the inferred fields are around 0.95 and 0.98, respectively. When the template becomes more complex, as in the Medium

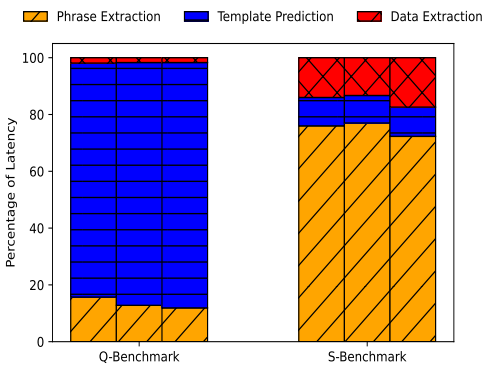


Fig. 11. Latency Breakdown.

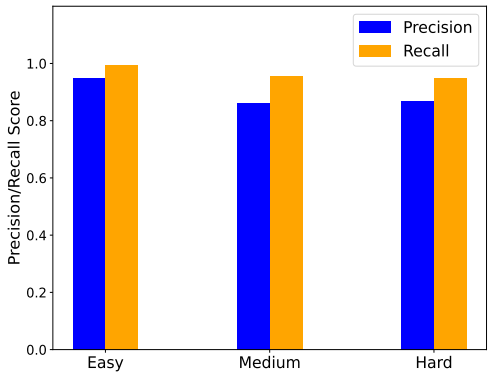


Fig. 12. Field Inference.

and Hard datasets, the precision drops to around 0.86 while the recall still remains close to 0.95. A high recall is important for predicting the template structure since the set of key rows will likely be recovered, which are the backbone of tables.

Template structure inference helps correct field inference errors. First, metadata (headers, footers, etc.) incorrectly inferred as keys or values are often corrected, as they rarely align with rows in table or key-value blocks, violating constraints in the row labeling problem. Second, false positives of fields are frequently corrected after row labeling. For example, the false positives in a value row will be corrected once the row is identified as a value row. These observations highlight TWIX's robustness in field inference.

Experiment 7: Error Pattern Analysis. We identify six common error types observed in the compared tools below and summarize them in Table 5, where we use ✓✓ and ✓ to indicate errors that *most significantly affect* and *moderately affect* each tool, respectively. .

Type (1): Column Misalignment. When multiple consecutive values are missing in a row in a table block, vision LLMs and Evaporate often fail to count the exact number of missing values, resulting in mismatched values and rows. **Type (2): Misidentification of Table Headers**. AzureDI fails to identify the table header correctly, instead selecting the first row as the header in 4 out of 34 datasets. Textract similarly misidentifies table headers, particularly in documents with complex layouts. **Type (3): Misidentification of Data Blocks**. All baselines struggle to accurately identify data blocks in complex layouts, such as in Medium and Hard datasets. They may miss data blocks entirely or merge multiple blocks into one. For example, vision-based LLMs show inconsistent behavior, occasionally extracting partial data or missing entire blocks. **Type (4): Phrase Extraction Errors from OCR**. OCR-based phrase extraction is imperfect and affects all tools. OCR may merge closely spaced phrases into one or split a long phrase into multiple parts. While such errors are relatively infrequent, they can still impact results, particularly when fields like table headers are extracted incorrectly. **Type (5): Metadata Misclassified as Fields or Values**. If a metadata row is misclassified as a value row and aligns with a table header, TWIX incorrectly includes it in the table block, causing false positives. Similar behavior is observed in other tools; vision-LLMs frequently create new fields for metadata and incorporate them into data blocks. **Type (6): Field Inference Errors**. None of the tools achieves perfect field inference, which affects downstream tasks such as table or key-value block inference in the baselines or row labeling in TWIX.

OCR Error Analysis. We describe below the three most common OCR errors, their impact on TWIX, and strategies to handle them. **1) Phrase Misspelling.** OCR may produce misspelled phrases (e.g., Date misread as Dafe). If the error occurs in a value, the effect is local and minimal. If it occurs in a field, the errors are *consistent* across records due to the shared template (e.g., all instances of "Date" are misread as "Dafe"). This consistency allows TWIX to correctly infer the template.

	Type 1	Type 2	Type 3	Type 4	Type 5	Type 6
Textract		✓✓	✓✓	✓	✓	✓✓
vLLM-S	✓✓		✓✓	✓	✓	✓
vLLM-C	✓✓		✓✓	✓	✓	✓
AzureDI		✓✓	✓✓	✓	✓	✓✓
Eva-D	✓		✓	✓	✓✓	✓✓
Eva-C	✓✓		✓✓	✓	✓	✓✓
TWIX				✓	✓	✓✓

Table 5. Frequent Error Patterns of Compared Tools (✓✓ indicates frequent, and ✓ indicates sometimes).

LLMs can be used during post-processing to semantically correct such errors in fields, though that is beyond the scope of TWIX. **2) Bounding Box Extraction Inaccuracy.** TWIX uses bounding boxes for row alignments when inferring template structure. Bounding boxes, if imprecise, are typically only slightly over-sized and are still reliable to determine vertical alignment. **3) Multi-row Phrase Extraction.** When a phrase spans multiple rows within a table cell, OCR may extract it as separate phrases. While improving OCR is beyond the scope of TWIX, multi-row phrases can be extracted by either using visual cues (e.g., cell borders or lines) to merge the fragments into a single phrase or applying semantic checks with LLMs to determine whether the split phrases should be merged.

8 Related Work

Our work is relevant to document and web data extraction, as well as document layout analysis.

Document Data Extraction. There have been multiple papers [14, 17, 52, 57] and industrial tools [1, 3, 4] aimed at extracting data from documents. Most extraction tools, e.g., [1, 3, 4, 14, 17, 52] are general-purpose solutions for extracting tables or entities from visually rich form-like documents. Evaporate [17] uses LLMs to generate scripts that are then used for data extraction. These scripts encode heuristic rules that are not robust for reliable extraction.

Tools from industry like Textract [1], Google Document AI [4], and Azure Document Intelligence [3] use pretrained models to extract structured data, such as tables or key-value pairs, from form-like documents. These tools perform well on simple layouts and domains well-represented in their training data (e.g., Google Document AI offers models for specific domains like taxes or invoices). However, they often fail on unseen documents with complex layouts and incur significant costs and latency. Recent advances in vision-based LLMs, such as GPT-4 Vision [6], show promise but lack consistent performance, with high latency and substantial costs.

Learning-based extraction [18, 35, 37, 42, 45, 46, 57, 61] retrieves values for user-specified fields from documents by training deep learning models on human-labeled data. However, it requires significant human effort (e.g., specifying fields and labels) and doesn't capture relationships between extracted field values, as discussed in Section 1. In contrast, our unsupervised approach uncovers the template—the backbone of templatized documents—and efficiently extracts structured data. For example, Parthasarathy et al. [48] introduce the concept of landmarks to narrow down the region of interest in a document and then extract values for user-specified fields, while TWIX preserves relationships in the extracted data.

Web Data Extraction. Our work is also related to web data extraction, which primarily relies on HTML tags [15, 16, 20, 23, 26, 28–33, 36, 40, 44, 47, 51, 52]. The earliest work in this vein analyzed differences between pages generated using the same HTML template to learn the template through techniques like regular expressions [16, 28, 36, 40]. Other papers extended this to develop robust wrappers to handle the evolution of underlying HTML templates in web pages over time [29, 47], while others explored generating domain-centric wrappers for web-scale information extraction, designed to tolerate noise [20, 30–32]. For example, Miria [26] extracts records from

websites by identifying invariants across records based on HTML tag trees. Cetorelli et al. [23] introduce a landmark-based grammar from a set of web pages with a common HTML template. To extract relations from semi-structured web data, Lockard et al. [52] proposed a distant supervision approach, while DeepDive [44] further leveraged XML and HTML-specific feature descriptors. However, HTML or XML tags present in web-pages or semistructured documents indicating nesting relationships are often not available in documents like PDFs.

Several studies extract tables from the web without relying on HTML tags. Chu et al. [27] split phrases in record rows into cells aligned with corresponding columns, while Chen et al. [25] extract information from spreadsheets by leveraging font formatting to extract metadata and using learning-based methods to label rows. Cafarella et al. [22] extract fact tuples by parsing natural language sentences on the web. Finally, Gao et al. [34] explore an unsupervised approach to extract structures for log datasets. These methods primarily focus on data with simple layouts or structures, like tables or logs, whereas our approach handles complex visual layouts, including nested combinations of tables and key-value structures.

Document Layout Analysis. Document layout analysis (DLA) [19, 21, 41, 50, 55, 56, 58, 60] detects various document layouts, such as pages, texts, tables, images, titles, headers, and footers, using visual features (e.g., font size and type), content, and structural patterns. While effective for coarse-grained components like text and table blocks, DLA struggles with fine-grained components, such as mixed key-value and table blocks. Combining DLA with our approach could potentially enhance data extraction by first detecting structured portions in long documents with text or images, allowing our method to handle the structured parts and expanding its applicability. This remains an interesting avenue for future work.

9 Conclusion

We present TWIX, a robust, efficient, and effective tool to extract structured data from a collection of templatized documents. TWIX first infers a flexible visual template used to create documents, using which it separates templatized documents into nested records and data blocks within records, enabling accurate and efficient data extraction from each block. We demonstrate hardness for the underlying problems, while also providing correctness guarantees. TWIX combines an optimization approach for principled template discovery, while leveraging LLMs to provide semantic knowledge in carefully targeted ways. Our experiments show that TWIX outperforms baselines significantly across accuracy, latency, and cost.

Acknowledgments

We acknowledge support from grants DGE-2243822, IIS-2129008, IIS-1940759, IIS-1955488, IIS-2027575, and IIS-1940757 awarded by the National Science Foundation, DOE award DE-SC0016260, AC02-05CH11231, DARPA Agreement No. HR00112590131, funds from the State of California, an NDSEG Fellowship, funds from the Alfred P. Sloan Foundation, as well as EPIC lab sponsors: Adobe, Bridgewater, Google, G-Research, Microsoft, PromptQL, Sigma Computing, and Snowflake. Compute credits were provided by Azure, Modal, NSF (via NAIRR), and OpenAI.

References

[1] 2024. *https://aws.amazon.com/textract/.*
[2] 2024. *https://azure.microsoft.com/en-us/products/ai-services/ai-document-intelligence.*
[3] 2024. *https://azure.microsoft.com/en-us/products/ai-services/ai-document-intelligence.*
[4] 2024. *https://cloud.google.com/document-ai.*
[5] 2024. *https://github.com/jsvine/pdfplumber.*
[6] 2024. *https://help.openai.com/en/articles/8555496-gpt-4-vision-api.*
[7] 2024. *https://openai.com/api/pricing/.*
[8] 2024. *https://pymupdf.readthedocs.io/en/latest/.*
[9] 2024. *https://www.gurobi.com/resources/ch4-linear-programming-with-python/.*
[10] 2025. *https://github.com/tesseract-ocr/tesseract.*
[11] 2025. *https://mistral.ai/news/mistral-ocr.*
[12] 2025. *https://www.ghx.com/media/02yp52j4/billentis-report-the-e-invoicing-journey-2019-2025.pdf.*
[13] 2025. *Technical Report of Visual Template Inference for Data Extraction from Documents; https://arxiv.org/abs/2501.06659.*
[14] Milan Aggarwal, Hiresh Gupta, Mausoom Sarkar, and Balaji Krishnamurthy. 2021. Form2Seq: A framework for higher-order form structure extraction. *arXiv preprint arXiv:2107.04419* (2021).
[15] Eugene Agichtein and Luis Gravano. 2000. Snowball: Extracting relations from large plain-text collections. In *Proceedings of the fifth ACM conference on Digital libraries*. 85–94.
[16] Arvind Arasu and Hector Garcia-Molina. 2003. Extracting structured data from web pages. In *Proceedings of the 2003 ACM SIGMOD international conference on Management of data*. 337–348.
[17] Simran Arora, Brandon Yang, Sabri Eyuboglu, Avanika Narayan, Andrew Hojel, Immanuel Trummer, and Christopher Ré. 2023. Language Models Enable Simple Systems for Generating Structured Views of Heterogeneous Data Lakes. *Proceedings of the VLDB Endowment* 17, 2 (2023), 92–105.
[18] Ting Bai, Ji-Rong Wen, Jun Zhang, and Wayne Xin Zhao. 2017. A neural collaborative filtering model with interaction-based neighborhood. In *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*. 1979–1982.
[19] Galal M Binmakhashen and Sabri A Mahmoud. 2019. Document layout analysis: a comprehensive survey. *ACM Computing Surveys (CSUR)* 52, 6 (2019), 1–36.
[20] Philip Bohannon, Nilesh Dalvi, Yuval Filmus, Nori Jacoby, Sathiya Keerthi, and Alok Kirpal. 2012. Automatic web-scale information extraction. In *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*. 609–612.
[21] Thomas M Breuel. 2003. High performance document layout analysis. In *Proceedings of the Symposium on Document Image Understanding Technology*, Vol. 5.
[22] Michael J Cafarella, Jayant Madhavan, and Alon Halevy. 2009. Web-scale extraction of structured data. *Acm Sigmod Record* 37, 4 (2009), 55–61.
[23] Valerio Cetorelli, Paolo Atzeni, Valter Crescenzi, and Franco Milicchio. 2021. The smallest extraction problem. *Proceedings of the VLDB Endowment* 14, 11 (2021), 2445–2458.
[24] Sarah E Chasins, Maria Mueller, and Rastislav Bodik. 2018. Rousillon: Scraping distributed hierarchical web data. In *Proceedings of the 31st Annual ACM Symposium on User Interface Software and Technology*. 963–975.
[25] Zhe Chen and Michael Cafarella. 2013. Automatic web spreadsheet data extraction. In *Proceedings of the 3rd International Workshop on Semantic Search over the Web*. 1–8.
[26] Zhijia Chen, Weiyi Meng, and Eduard Dragut. 2022. Web record extraction with Invariants. *Proceedings of the VLDB Endowment* 16, 4 (2022), 959–972.
[27] Xu Chu, Yeye He, Kaushik Chakrabarti, and Kris Ganjam. 2015. Tegra: Table extraction by global record alignment. In *Proceedings of the 2015 ACM SIGMOD international conference on management of data*. 1713–1728.
[28] Valter Crescenzi and Giansalvatore Mecca. 2004. Automatic information extraction from large websites. *Journal of the ACM (JACM)* 51, 5 (2004), 731–779.
[29] Nilesh Dalvi, Philip Bohannon, and Fei Sha. 2009. Robust web extraction: an approach based on a probabilistic tree-edit model. In *Proceedings of the 2009 ACM SIGMOD International Conference on Management of data*. 335–348.
[30] Nilesh Dalvi, Ravi Kumar, Bo Pang, Raghu Ramakrishnan, Andrew Tomkins, Philip Bohannon, Sathiya Keerthi, and Srujana Merugu. 2009. A web of concepts. In *Proceedings of the twenty-eighth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*. 1–12.
[31] Nilesh Dalvi, Ravi Kumar, and Mohamed Soliman. 2011. Automatic wrappers for large scale web extraction. *arXiv preprint arXiv:1103.2406* (2011).
[32] Nilesh Dalvi, Ashwin Machanavajjhala, and Bo Pang. 2012. An analysis of structured data on the web. *arXiv preprint arXiv:1203.6406* (2012).

[33] Oren Etzioni, Michael Cafarella, Doug Downey, Stanley Kok, Ana-Maria Popescu, Tal Shaked, Stephen Soderland, Daniel S Weld, and Alexander Yates. 2004. Web-scale information extraction in knowitall: (preliminary results). In *Proceedings of the 13th international conference on World Wide Web*. 100–110.

[34] Yihan Gao, Silu Huang, and Aditya Parameswaran. 2018. Navigating the data lake with datamaran: Automatically extracting structure from log datasets. In *Proceedings of the 2018 International Conference on Management of Data*. 943–958.

[35] Anoop Raveendra Katti, Christian Reisswig, Cordula Guder, Sebastian Brarda, Steffen Bickel, Johannes Höhne, and Jean Baptiste Faddoul. 2018. Chargrid: Towards understanding 2d documents. *arXiv preprint arXiv:1809.08799* (2018).

[36] Mohammed Kayed and Chia-Hui Chang. 2009. FiVaTech: Page-level web data extraction from template pages. *IEEE transactions on knowledge and data engineering* 22, 2 (2009), 249–263.

[37] Vu Le and Sumit Gulwani. 2014. Flashextract: A framework for data extraction by examples. In *Proceedings of the 35th ACM SIGPLAN Conference on Programming Language Design and Implementation*. 542–553.

[38] Yiming Lin, Madelon Hulsebos, Ruiying Ma, Shreya Shankar, Sepanta Zeigham, Aditya G Parameswaran, and Eugene Wu. 2024. Towards accurate and efficient document analytics with large language models. *arXiv preprint arXiv:2405.04674* (2024).

[39] Chunwei Liu, Matthew Russo, Michael Cafarella, Lei Cao, Peter Baille Chen, Zui Chen, Michael Franklin, Tim Kraska, Samuel Madden, and Gerardo Vitagliano. 2024. A declarative system for optimizing ai workloads. *arXiv preprint arXiv:2405.14696* (2024).

[40] Wei Liu, Xiaofeng Meng, and Weiyi Meng. 2009. Vide: A vision-based approach for deep web data extraction. *IEEE transactions on knowledge and data engineering* 22, 3 (2009), 447–460.

[41] Shangbang Long, Siyang Qin, Dmitry Panteleev, Alessandro Bissacco, Yasuhisa Fujii, and Michalis Raptis. 2022. Towards end-to-end unified scene text detection and layout analysis. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 1049–1059.

[42] Navonil Majumder, Soujanya Poria, Alexander Gelbukh, and Erik Cambria. 2017. Deep learning-based document modeling for personality detection from text. *IEEE intelligent systems* 32, 2 (2017), 74–79.

[43] Oshri Naparstek, Ophir Azulai, Inbar Shapira, Elad Amrani, Yevgeny Yaroker, Yevgeny Burshtein, Roi Pony, Nadav Rubinstein, Foad Abo Dahood, Orit Prince, et al. 2024. KVP10k: A Comprehensive Dataset for Key-Value Pair Extraction in Business Documents. In *International Conference on Document Analysis and Recognition*. Springer, 97–116.

[44] Feng Niu, Che Zhang, Christopher Ré, and Jude W Shavlik. 2012. DeepDive: Web-scale Knowledge-base Construction using Statistical Learning and Inference. *VLDS* 12 (2012), 25–28.

[45] Nerya Or and Shlomo Urbach. 2021. Few-shot learning for structured information extraction from form-like documents using a diff algorithm. In *Proc. Document Intell. Workshop KDD*.

[46] Shubham Singh Paliwal, D Vishwanath, Rohit Rahul, Monika Sharma, and Lovekesh Vig. 2019. Tablenet: Deep learning model for end-to-end table detection and tabular data extraction from scanned document images. In *2019 International Conference on Document Analysis and Recognition (ICDAR)*. IEEE, 128–133.

[47] Aditya Parameswaran, Nilesh Dalvi, Hector Garcia-Molina, and Rajeev Rastogi. 2011. Optimal schemes for robust web extraction. In *Proceedings of the VLDB Conference*, Vol. 4. VLDB Endowment.

[48] Suresh Parthasarathy, Lincy Pattanaik, Anirudh Khatry, Arun Iyer, Arjun Radhakrishna, Sriram K Rajamani, and Mohammad Raza. 2022. Landmarks and regions: a robust approach to data extraction. In *Proceedings of the 43rd ACM SIGPLAN International Conference on Programming Language Design and Implementation*. 993–1009.

[49] Liana Patel, Siddharth Jha, Carlos Guestrin, and Matei Zaharia. 2024. Lotus: Enabling semantic queries with llms over tables of unstructured and structured data. *arXiv preprint arXiv:2407.11418* (2024).

[50] Akshay Gadi Patil, Omri Ben-Eliezer, Or Perel, and Hadar Averbuch-Elor. 2020. Read: Recursive autoencoders for document layout generation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*. 544–545.

[51] Sunita Sarawagi et al. 2008. Information extraction. *Foundations and Trends® in Databases* 1, 3 (2008), 261–377.

[52] Ritesh Sarkhel and Arnab Nandi. 2021. Improving information extraction from visually rich documents using visual span representations. *Proceedings of the VLDB Endowment* 14, 5 (2021).

[53] Shreya Shankar, Tristan Chambers, Tarak Shah, Aditya G Parameswaran, and Eugene Wu. 2024. DocETL: Agentic Query Rewriting and Evaluation for Complex Document Processing. *arXiv preprint arXiv:2410.12189* (2024).

[54] Štěpán Šimsa, Milan Šulc, Michal Uřičář, Yash Patel, Ahmed Hamdi, Matěj Kocián, Matyáš Skalický, Jiří Matas, Antoine Doucet, Mickaël Coustaty, et al. 2023. Docile benchmark for document information localization and extraction. In *International Conference on Document Analysis and Recognition*. Springer, 147–166.

[55] Carlos Soto and Shinjae Yoo. 2019. Visual detection with context for document layout analysis. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*. 3464–3470.

[56] Zineng Tang, Ziyi Yang, Guoxin Wang, Yuwei Fang, Yang Liu, Chenguang Zhu, Michael Zeng, Cha Zhang, and Mohit Bansal. 2023. Unifying vision, text, and layout for universal document processing. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 19254–19264.

[57] Sandeep Tata, Navneet Potti, James B Wendt, Lauro Beltrao Costa, Marc Najork, and Beliz Gunel. 2021. Glean: Structured extractions from templatic documents. *Proceedings of the VLDB Endowment* 14, 6 (2021), 997–1005.

[58] Yiheng Xu, Minghao Li, Lei Cui, Shaohan Huang, Furu Wei, and Ming Zhou. 2020. Layoutlm: Pre-training of text and layout for document image understanding. In *Proceedings of the 26th ACM SIGKDD international conference on knowledge discovery & data mining*. 1192–1200.

[59] Yiheng Xu, Tengchao Lv, Lei Cui, Guoxin Wang, Yijuan Lu, Dinei Florencio, Cha Zhang, and Furu Wei. 2022. XFUND: a benchmark dataset for multilingual visually rich form understanding. In *Findings of the Association for Computational Linguistics: ACL 2022*. 3214–3224.

[60] Xiao Yang, Ersin Yumer, Paul Asente, Mike Kraley, Daniel Kifer, and C Lee Giles. 2017. Learning to extract semantic structure from documents using multimodal fully convolutional neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 5315–5324.

[61] Yang Yang, Zhilei Wu, Yuexiang Yang, Shuangshuang Lian, Fengjie Guo, and Zhiwei Wang. 2022. A survey of information extraction based on deep learning. *Applied Sciences* 12, 19 (2022), 9691.

Received April 2025; revised July 2025; accepted August 2025