VRDSynth: Synthesizing Programs for Visually Rich Document Information Extraction

Anonymous Author(s)

ABSTRACT

BachLe: 1: Elaborate in one or two sentences to argue that VRD is commonly used and important. We introduce a program synthesis method to automatically generate programs to extract relations from visually rich documents. This works in common situations where companies have to extract structured information from medical records, bills, purchase receipts, and insurance forms from different vendors. Unlike existing tools that work on semi-structured inputs such as spreadsheets or webpages, or on specific formats, our method takes as input a document graph constructed from scanned visually rich documents and synthesizes programs that link semantic entities from scanned documents in different formats and domains. Toward this goal, we propose a new domain-specific language (DSL) to capture the spatial and textual relations between document entities. To navigate the search space of this DSL, we also present a novel synthesis algorithm, namely, VRDSynth. This algorithm makes use of three elements: (1) frequent spatial relations between entities to construct initial programs, (2) equivalent reduction to prune the search space, and (3) a combination of positive, negative, and mutually exclusive programs to improve the coverage of programs.

We evaluate our method on two popular visually rich document understanding benchmarks - FUNSD and XFUND in the semantic entity linking task, consisting of 1,600 forms in 8 different languages. In this benchmark, VRDSynth's synthesized programs outperform the pre-trained LayoutXLM in 6 out of 8 tasks despite having no prior training data. This is largely due to the better precision of the synthesized programs. Noticeably, in FUNSD, VRDSynth achieved an improvement of 35% over LayoutXLM in terms of F1 score. Regarding efficiency, VRDSynth performs similarly with LayoutXLM in the same settings despite the speed differences between the implemented languages (Python vs C++). Furthermore, while LayoutXLM requires 1.48GB to store and 3GB of memory for inferences, VRDSynth requires only 1MB and 380MB to store and inference, respectively.

ACM Reference Format:

Anonymous Author(s). 2023. VRDSynth: Synthesizing Programs for Visually Rich Document Information Extraction. In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

Conference'17, July 2017, Washington, DC, USA

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Businesses have to collect and store information from varied sources for administrative tasks: from medical records, bills, purchase receipts, and insurance forms to technical reports from different vendors. These administrative documents are often characterized by a mix of different visual elements such as images, graphics, tables, diagrams, etc, hence, they are also called visually-rich documents (VRDs) [1]. Since these documents are the main form of retaining and transferring information between business enterprises, a massive amount of data today is stored in the form of VRDs. It is essential for key information to be extracted before being analyzed, thus, various deep learning-based and heuristic techniques have been proposed [2–12].

To deal with varied combinations of layout, visual and textual features, deep learning techniques either leverage a pre-trained transformer model [2–4], a segmentation model on a new representation [8] or a node-classification on a graph-based representation [5] for semantic entity recognition (i.e., identifying which words belong to some specific desired categories). To store and query information from this semi-structured output, FUNSD [13] proposed the semantic entity linking problem: given the recognized entity, identify which values belong to which key. while state-of-the-art works [4, 5, 14] managed to identify which words are key and which are value, semantic linking between these key-and-values is still unaddressed [13]. In detail, on FUNSD [13], BERT-based entity linking achieved F1 score of only 0.04, which consists of 0.99 recall and 0.02 precision score, meaning that the approach can identify most of the link, but the majority of them are false positives. While LayoutXLM reported promising results of fairly accurate linking [14] of 0.7-0.8 F1 scores, it has been shown that actual running only gives 0.5-0.6 F1, given the settings that the training model cannot see the validation during training [15]. This can be attributed to the difficulty in applying modeling domain-specific constraints to these models. On the other hand, program synthesis techniques, which make use of domain-specific constraints by expert, are capable of synthesizing highly precise programs in data wrangling and information extraction, given a limited amount of data [1, 16–18].

BachLe: 2: Please elaborate more in one paragraph here about how unsatisfactory the current approaches are and what are the underlying reasons. Then move on to motivate your approach.

In this paper, we propose a synthesis-based approach to generate programs that link semantic entities in visually rich documents. First, we construct the symbolic visually rich document, consisting of a collection of entities and spatial relations between them. Each entity consists of a bounding box location, the pre-recognized label from the semantic entity recognition step, and its textual content. The relation is built based on spatial alignment (top, down, left, right) between entities. This domain is different from the existing synthesis works that operate on spreadsheets [16, 17, 19], sequence [1]. Instead, this domain is closer to that of LRSyn [20],

however, LRSyn focuses on extracting values directly from domain-specific forms while we aim for general extraction of entities relation based on the output of the semantic entity recognition step, which has been show to achieve high accuracy in existing literature. Following that, we propose a novel DSL that allows programs to capture these spatial and textual relations between entities: each program in this domain describes a linking program that locates all pair of entities that matches structural and textual constraint to be linked together. Finally, to synthesize the program in this DSL, we propose VRDSynth, a novel synthesis algorithm. VRDSynth works in two phases: identifying initial programs and iteratively refining programs. In the first stage, VRDSynth mines the frequented relation between semantically linked entities from the training documents and uses these frequented relations to construct initial programs that filter out potential pairs of semantically linkable entities. Since these programs can be imprecise, VRDSynth further refine these programs by searching for textual, label, position, or relation constraints that result in more precise programs. During this search, VRDSynth makes use of equivalent reduction [21] to prune the search space. At the end of each iterative refinement step, VRDSynth combines positive, negative, and mutually exclusive programs to produce more precise programs. Put replication package

BachLe: 3: The figure 1 doesn't show anything useful for reader to understand what's going on. Make the figure clearer, add caption to explain what's happening in the figure.

We evaluate VRDSynth on the semantic entity linking benchmark from FUNSD and XFUND [13, 22], these two datasets together consist of 1,600 forms in 8 languages - English, German, French, Italian, Japanese, Portuguese, and Chinese. On this benchmark, we show that, even when requiring no prior pre-training data, VRDSynth's managed to synthesize programs that outperform LayoutXLM in terms of F1 score on 6 out of 8 languages. In detail, these programs have consistently higher precision than LayoutXLM (0.693 versus 0.429) on average. Notably, in English, the synthesized programs' F1 score is 34.9% higher than the LayoutXLM. In terms of efficiency, the programs perform similarly to LayoutXLM despite being implemented in Python instead of C++ while requiring significantly less memory to store (1MB vs 1.48GB) and to run (380MB vs 3.04GB). We summarize the main contribution as follows:

- We present VRDSynth framework that bridges the gap between deep learning and program language for synthesizing programs to extract semantic entity relations from visually rich documents.
- We propose a novel domain-specific language (DSL) and a novel synthesis algorithm that efficiently navigates the DSL's program space by pruning and sketching, in which sketches are obtained by mining patterns from training data and pruning is achieved by equivalence reduction and combinations of negative and mutually-exclusive programs.
- We perform extensive evaluation and show that VRDSynth's synthesized programs outperform the pre-trained language model while being as efficient, as well as the contribution of the chosen components.

BachLe: 4: Add a summary of sections here. The rest of the paper is organized as follows. Section 2 describes the related works on visually-rich document processing. Section 3 describes our settings and problem formulation. Section 4 describes our domain-specific language. Section 7 shows our experiment settings and results. We conclude the paper in Section 8.

2 RELATED WORKS

In handling varied layout documents, there are three major directions: rule-based approach, deep learning-based approach, and program synthesis.

Rule-based and template-based approach Traditional approaches in document image processing makes use of heuristic rules [11, 23–25]. These rules rely on carefully designed feature engineering (projection profile alignments [11], similar words and box size and distances [24, 25] to identify the desired document entities [26]. Based on these alignments, researchers [24, 27] have shown that a template can also be built for each document form [28]. After building the template, visual and textual features can be used to calculate the similarity between a given template and a target document image, e.g., [27, 29]. Additionally, [30] proposes a finer-grained template using the combinations of TF-IDF, angle, and distance between entities. While the rule-based and template-based methods perform well for a fixed set of document formats, when there exist more varied document formats from different business vendors, the management of these templates can become intractable. Additionally, template-based approaches are also prone to varying visual quality of documents, i.e., document images with low visual quality may unduly be classified as new templates, rendering template-based approaches imprecise. We aim to automatically aid developers in developing information extraction rules by automatically synthesizing programs based on example input document annotation. The synthesized rules can be seen as a matching template for each entity.

Deep Learning-based Information Extraction More recently, deep learning has also been applied for information extraction from visually rich documents: [10] uses a recurrent neural network (RNN) to extract entities based on their textual features, they ignored the contribution of image and layout features and relied on the left-to-right reading order. [9] proposed a new chargrid representation and used a convolutional neural network (CNN) to segment the regions containing desired entities in the chargrid. Nevertheless, 2D chargrid or segmentation-based IE methods usually do not work in the case of dense documents with little space between entities. Graph Convolution Network (GCN) [31] [32] have also been applied for visually-rich document information extraction by converting the input document into a document graph [5, 33–37] and uses graph neural networks for entity recognition by formulating the problem into node classification, in which each entity is a node. Our work relies on the output of semantic entity recognition by these works, furthermore, we also reuse the document graph as the input domain and propose a new DSL along with a synthesis algorithm on this domain.

Program Synthesis for Information Extraction Program synthesis has long been used for different scenarios, ranging from programs handling primitive operations [18, 38], extracting information from spreadsheets [16, 17, 19], sequences [1], web data, PDF documents [7, 20, 39] and manipulating images [40]. These works generally propose either a new domain-specific language or

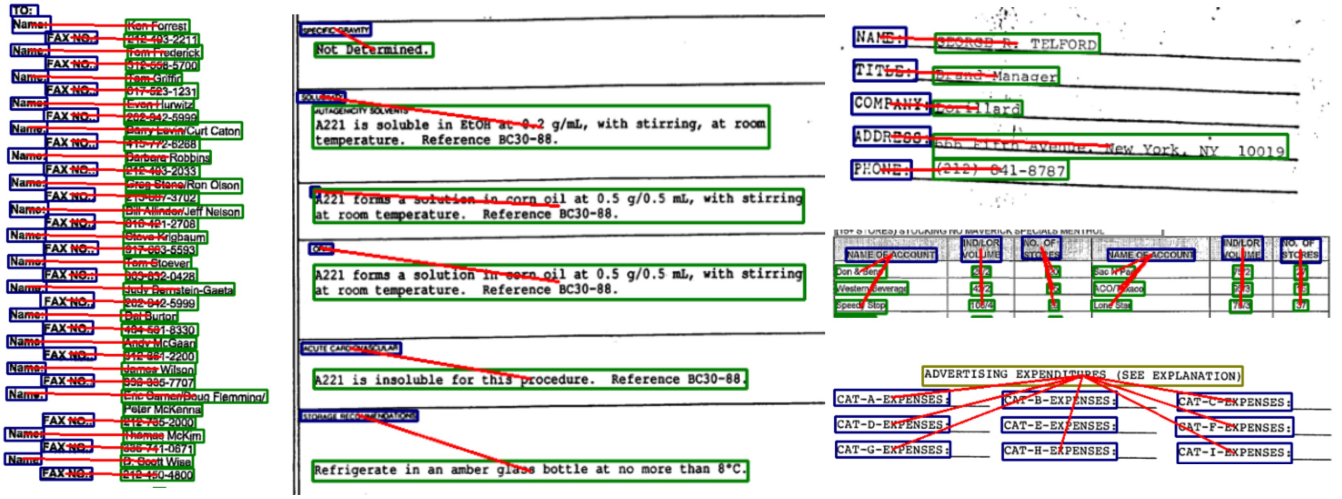


Figure 1: Example output of VRDSynth's synthesized programs in different layout. The programs collectively link each corresponding key (e.g., names, fax) to corresponding values or headers to corresponding keys.

a new technique for efficiently navigating the search space in synthesizing programs. The closest works to ours are that of [1, 7, 20] and [40]. [7] and [20] propose the usage of a transition system to extract the desired entity from a starting entity that matches specific conditions: containing a word, or lies on the left or right of a specific block. [40] proposed using a new DSL that involves the relation between entities to localize the desired image regions. Our domain-specific language also contains elements specifying the spatial relation between entities as well as textual constraint (containing specific text or specific character) like [7, 20]. However, we also allow a more flexible scheme in specifying structural constraints between elements: while [7] and [20] only allow specifying a path, we allow specifying a graph as a relation constraint between entities. The DSL also allows interaction between programs by union and exclusion of synthesized program output.

3 BACKGROUND

3.1 Symbolic Representation of Visually Rich Documents

We give a brief definition of a symbolic representation of a document D_V below, following [5].

Definition 1 (Symbolic Visually Rich Document)). *A document D_V is conceptualized as an ensemble of elements denoted by $D_V = \{e_1, e_2, \dots e_N\}$, where N is the total number of elements in the documents. Each element e_i is a tuple of (t_i, p_i, l_i) where t_i is the textual content of the element, p_i is characterized by coordinates $(x_{i,0}, y_{i,0}, x_{i,1}, y_{i,1})$ denotes the rectangular bounds of the element, originating from the starting coordinate $(x_{i,0}, y_{i,0})$ to its conclusive boundary at $(x_{i,1}, y_{i,1})$. Finally, l_i is the predicted label of each element using DL models.*

To facilitate the ground for synthesizing relational rules on these visually rich documents, we also build a set of relations $D_R = \{(e_i, e_j)\}$. These relations consist of the basic spatial relation between elements such as *top, down, left, right* nearest elements, constructed by comparing each element's bounding box positions.

3.2 Problem Definition

Given this definition of symbolic representation of symbolic visually rich document, we define the task of semantic entity linking in Definition 2.

Definition 2 (Semantic Entity Linking). *Given a Symbolic VRD D as defined in Definition 1, the process of Semantic Entity Linking involves establishing connections, $\mathcal{L}$, between pairs of elements $(e_i, e_j) \in \mathcal{E} \times \mathcal{E}$ such that e_j is either the key, value or sub-key of e_i*

In the context of visually rich documents, the "connection" in Semantic Entity Linking refers to the relational information embedded within the document elements. These connections are not spatial or sequential linkages but are semantic associations that define the hierarchy of information in the document.

For instance, consider a form in Figure 2: the entity "Event Name" is the question, and "The USQ Spring Festival" is the answer. Similarly, in the same form, a section titled "Special Event Information Sheet" may have sub-sections (e.g., "General Information"), which are semantically linked as they elaborate on the broader "Special Event Information Sheet" category. The problem of identifying these relationships is called "Semantic Entity Linking".

To formulate the input and output of program synthesis, we present a formalization for semantic entity linking in the context of program synthesis. Suppose we have a dataset consisting of instances of symbolic visually rich documents, each represented as D_i where $D_i = \{e_1, e_2, \dots e_N\}$ are elements defined in Definition 1, alongside with these documents, we are provided with the grouping of entities $G_i = \{g_1, g_2 \dots g_K\}$ and the semantic link between L_i existing between elements, where each semantic link $(g_j, g_k) \in L_i$ is an ordered pair of semantic group from D_i, indicating a directional relationship from one element to another (i.e., $L_i \subseteq G_i \times G_i$). The aim of semantic entity linking is to synthesize programs that can

accurately identify the relationships between different elements within these documents. The formal requirement for the program synthesis can be expressed as follows:

$$\forall((g_1, g_2) \in L_i)\exists(P \in \mathcal{S}) \text{ s.t. } g_1 \in P(g_2) \text{ and } (P_{g_2}, g_2) \in L_i \forall g_2 \tag{1}$$

Intuitively, this means that for every relation in the document, there should exist a program that is capable of extracting relation between these entities if any.

To give a consistent input to program synthesis, we encode both specifications in the form of a list of pairs that are linked towards each other $M_i = \{(e_1, e_2)\}$. In the case of semantic entity grouping, we input pair-wise connections between each word. In the case of semantic entity linking, M_i would instead consist of all pairs of question-answer or header-question/header-sub headers. Note that the specifications only made use of the symbolic visually rich documents, while the relation between elements in the document will later be used for the domain-specific language programs.

3.3 Syntax-Guided Program Synthesis

Syntax-guided program synthesis (SyGuS) is a specific approach towards program synthesis, focusing on the generation of programs from a combination of a logical specification and a syntactic template. The logical specification captures the desired behavior of the to-be-synthesized program, often expressed in terms of a formal logic (e.g., first-order logic or propositional logic), while the syntactic template defines the space of allowable implementations, serving as a skeleton for the structure of the resultant program.

Our work is also related to SyGuS, as we also perform program searching and extension based on domain-specific grammar: First, we define a grammar that reflects the constructs necessary for our domain in Section 4. Furthermore, based on our goal, we impose certain constraints to better prune the search space with respect to the goal of finding more general programs.

4 DOMAIN SPECIFIC LANGUAGE

4.1 Grammar

P	:=	$\emptyset \mid \textbf{Union}(\{P\}) \mid \textbf{Exclude}(P, P) \mid$ $\textbf{Find}(\{V_{set}\}, \{R_{set}\}, C, \{V_{set}\})$
C	:=	$\textbf{And}(C, C) \mid \textbf{Or}(C, C) \mid \textbf{Not}(C) \mid PC$
PC	::=	$\textbf{Rel}(V, R, V) \mid LblC \mid StrC \mid FloatC \mid \textbf{True} \mid \textbf{False}$
$LblC$	::=	$VLabel == VLabel \mid RLabel == RLabel$
$StrC$	::=	$\textbf{Eq}(Str, Str) \mid \textbf{Contain}(Str, Str)$
$FloatC$	::=	$F < F \mid F > F$
$VLbl$	::=	$V.\textbf{label} \mid \textbf{Header} \mid \textbf{Question} \mid \textbf{Answer}$
$RLbl$	::=	$R.\textbf{label} \mid \mathbf{L}_0 \mid \mathbf{L}_1 \mid \ldots \mid \mathbf{L}_K$
Str	::=	$V.\textbf{text} \mid \textbf{"."} \mid \textbf{"/"} \mid \textbf{":"} \mid \textbf{"-"}$
F	::=	$RFloatProp \mid VFloatProp \mid FConst$
$RFloatProp$	::=	$R.\textbf{mag} \mid R.\textbf{proj0} \mid R.\textbf{proj1} \mid \ldots R.\textbf{projK}$
$VFloatProp$	::=	$V.\mathbf{x0} \mid V.\mathbf{x1} \mid V.\mathbf{y0} \mid V.\mathbf{y1}$
V_{set}	::=	$V \mid V, V_{set}$
R_{set}	::=	$R \mid R, R_{set}$
V	::=	$\mathbf{v_0} \mid \mathbf{v_1} \mid \ldots \mid \mathbf{v_{10}}$
R	::=	$\mathbf{r_0} \mid \mathbf{r_2} \mid \ldots \mid \mathbf{r_{10}}$
F	::=	$\mathbf{0} \mid \mathbf{0.1} \mid \mathbf{0.2} \mid \ldots \mid \mathbf{1.0}$

In the domain of symbolic visually-rich documents, each element bears significant semantic and relational information, the grammar is engineered to enable sophisticated operations and queries.

DSL Syntax. Our document analysis DSL, outlined in the preceding section, is engineered to embrace a wide spectrum of queries and manipulations pertinent to visually rich documents. At its core, the program is constructed from a set of guarded commands of the form P, where each command represents a specific operation like **Find** or **Exclude**, and the guard, represented as a condition C, designates the criteria of element attributes or relationships to apply these operations.

Programs and Operations (P): At the highest level, programs are composed using operations such as 'Union', 'Exclude', and 'Find', each serving distinct purposes. 'Union' amalgamates results from multiple queries, 'Exclude' filters out specific elements, and 'Find' executes a search based on criteria across visual and relational properties. Specifically, the **Find** programs, take as input 4 arguments: The list of symbolic entity variables (i.e., first V_{set}), the list of relation variables (i.e., the first R_{set}), the condition to filter these C, and the return set (i.e., the second V_{set} or we would call V_{set_return}). Given sets of potential elements and relationships (V_{set} and R_{set}, respectively), and a condition C to satisfy, it explores all possible bindings of these sets within the context of a document. However, the subtlety lies in how we determine the results from this operation, particularly those that belong to V_{set_return}. Here is a stepwise breakdown of the process:

- **Binding Exploration** Initially, the Find function orchestrates a comprehensive search where it explores all possible bindings of elements from V_{set} and relationships from R_{set} within the entire space of the document.
- **Condition Verification** Among the found set, **Find** filters the unique cases where the condition C holds true.
- **Resulting Set Accumulation** Finally, among these bindings **Find** collects the valuations of variables that belong to V_{set_return}.

Conditions C are recursively defined, grounding on base cases and complex constructs involving logical operations (**And**, **Or**, **Not**) and property comparisons within elements and their relationships, such as **Rel** for relationship properties, or direct comparisons of labels, strings, and numerical attributes. Since VRDs consist of both layout and textual information, the predicates and conditions PC, such as $LblC$ for label comparisons, $FloatC$ for floating-point comparisons to incorporate layout features, $StrC$ for string comparisons.

The label constraints $LblC$ make use of either the predicted word labels or relation labels to filter the possible valuation of entities V and relations R. These constraints dictate which label each binding entity should have.

The float constraints $FloatC$ perform comparisons between float properties of entities and relations.

The string constraints, $StrC$, focus on the textual content within the visually rich documents. These constraints allow for direct comparison of text strings, encompassing operations like checking equality or verifying if a particular substring is contained within an element's text. This part of the DSL is particularly crucial for

searching for elements containing certain keywords, phrases, or syntactic patterns.

The V and R variables represent entities and relationships in the document, respectively. Each entity can be associated with various properties, including labels ($VLbl$), textual content (Str), and numerical attributes (F), enabling a wide range of queries and operations based on these characteristics. Relationships, denoted by R, describe the connections or relative orientations between different entities, providing crucial context for understanding the document's structure and semantics.

The DSL also specifies sets of these entities and relationships (V_{set} and R_{set}), allowing operations to be performed over groups of items. These sets enable batch processing of elements, such as selecting multiple entities that meet certain criteria or applying transformations to several related elements simultaneously.

4.2 Denotational Semantics of the DSL

Semantic Domain Let us define the following domains: (1) D_P is the domain of all possible programs, (2) D_C is the domain of conditions D_V is the domain of visual elements in a document, (3) D_R is the domain of relationships between visual elements, (4) D_{Set} is the domain of sets of visual elements and (5) D_{Bool} is the domain of boolean values {`true`, `false`}.

Semantic Rules for Programs The interpretation of programs is defined recursively based on the structure of the program. Below we denote the semantics of the DSL programs. The semantic of **Find** relies on evaluating the potential bindings of (e.g., possible elements that can be assigned to) each relation and variable. Thus, we define *valuation function pair* $\langle b_v, b_r \rangle$ as a function that assigns values to variables and relationships:

$$b_v : V \to D_V \quad b_r : R \to D_R \tag{2}$$

Let B be the set of all possible bindings, where a single binding $b = \langle b_v, b_r \rangle$ is a function pair that assigns every variable in V or relation in R to a corresponding value or relation, such that for each $v \in V$ and $r \in R$, $b_v(v) \in D_V$ and $b_r(r) \in D_R$. We denote the valuation of v given binding b as $[\![v]\!]_b$. Moreover, the semantic function $[\![\cdot]\!]_b$ maps syntactic constructs (e.g., conditions) to their semantic (e.g., truth values) under a binding b. For a condition C, $[\![C]\!]_b$ is `true` or `false` depending on whether C holds under b.

$$[\![\textbf{Union}(P1, P2)]\!] = [\![P1]\!] \cup [\![P2]\!] \tag{3}$$

$$[\![\textbf{Exclude}(P1, P2)]\!] = [\![P1]\!] \setminus [\![P2]\!] \tag{4}$$

$$[\![\textbf{Find}(V_{set}, R_{set}, C, V_{set\_return})]\!] = \bigcup \{\textbf{result}(b) | b \in B\} \tag{5}$$

Given the definition of all possible bindings, the **Find** operation can be described as operating exhaustively over this set. Where $\textbf{result}(b)$ is the set of values computed by applying the **Find** operation within the context of specific binding b, defined as:

$$\textbf{result}(b) = \left\{ [\![v]\!]_b \;\middle|\; \begin{array}{l} [\![C]\!]_b = \texttt{true} \\ v \in V_{set\_return} \end{array} \right\} \tag{6}$$

The notation $[\![C]\!]_b$ represents the evaluation of condition C given b as the binding. We list the semantics of each rule below.

Semantic Rules for Conditions Conditions evaluate to boolean values based on the attributes of visual elements and relationships. We interpret conditions as functions from visual elements and relationships to boolean values.

$$[\![\textbf{And}(C_1, C_2)]\!]_b = [\![C_1]\!]_b \wedge [\![C_2]\!]_b \tag{7}$$

$$[\![\textbf{Or}(C_1, C_2)]\!]_b = [\![C_1]\!]_b \vee [\![C_2]\!]_b \tag{8}$$

$$[\![\textbf{Not}(C)]\!]_b = \textbf{not}([\![C]\!]_b) \tag{9}$$

$$[\![\textbf{Rel}(V_1, R, V_2)]\!]_b = \exists V_1 \in b, \exists V_2 \in b, \exists R \in b \tag{10}$$

$$[\![VLabel_1 == VLabel_2]\!]_b = ([\![VLabel_1]\!]_b == [\![VLabel_2]\!]_b) \tag{11}$$

$$[\![RLabel_1 == RLabel_2]\!]_b = ([\![RLabel_1]\!]_b == [\![RLabel_2]\!]_b) \tag{12}$$

$$[\![F_1 < F_2]\!]_b = ([\![F_1]\!]_b < [\![F_2]\!]_b) \tag{13}$$

$$[\![Str_1 == Str_2]\!]_b = [\![Str_1]\!]_b == [\![Str_2]\!]_b \tag{14}$$

$$[\![\textbf{Contains}(Str_1, Str_2)]\!]_b = [\![Str_1]\!]_b \supseteq [\![Str_2]\!]_b \tag{15}$$

Visual and relationship semantics

$$[\![V]\!]_b = b(V) \tag{16}$$

$$[\![R]\!]_b = b(R) \tag{17}$$

Intuitively, a **Find** program would gather all possible bindings of each variable and relations in V_{set} and R_{set} and identify the bindings that match the condition C. After obtaining the set of valid bindings, **Find** aggregates the valuation of variables belonging to the V_{set_return}. We give demonstrations on an example of Find Program in both the case of semantic entity grouping and semantic entity linking below.

5 MOTIVATING EXAMPLE

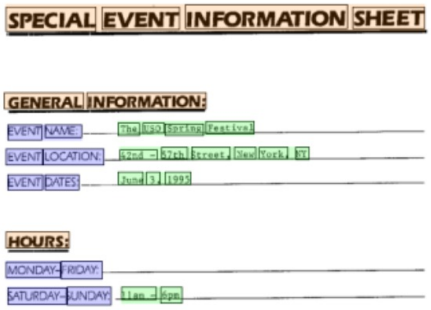
In many real-world scenarios involving Visually-Rich Documents (VRDs), there is a need to identify and group entities that relate to a specific target. For instance, consider an administrative form where multiple fields pertain to a specific category, like 'Event Name'. Given the target word 'Event', one might be interested in finding all related terms that belong to the same 'Event Name', ' Event Location', and 'Event Date'entity.

To provide a clear understanding of how one might tackle this challenge programmatically, we introduce a program depicted in Figure 2. This Find program represents a query or a search within a certain context or database, specifically looking for certain objects v_0 and v_1 and relations $r0$ that satisfy a set of conditions:

The first parameter $\{v_0, v_1\}$ dictates that the set of variables that this program operates on are v_0 and v_1. The second parameter dictates that there exists one relation r_1 that this **Find** program operates on. Followed by that, the following condition:

$$\begin{array}{l} \textbf{And}(\textbf{Rel}(v_0, r_0, v_1), \\ \quad \textbf{And}(r0.\textbf{label} == \textbf{right}, \\ \quad \textbf{And}(r0.\textbf{mag} < 0.1, v0.\textbf{label} == v1.\textbf{label}))) \end{array}$$

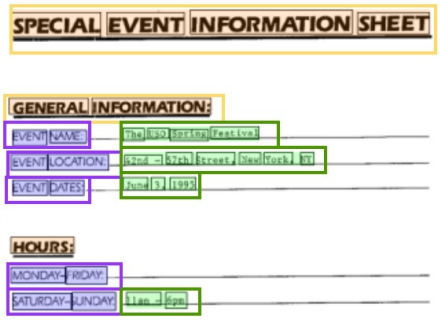
Requires that (1) there exists relation r_0 between v_0 and v_1, furthermore this relation has to have the label **right** (note that, we have replaced the label L_1 with **right** for ease of interpretation. Next, the *magnitude* property of relation r_0 is less than 0.1, this represents the spatial distance between the valuations of v_0 and v_1 has to be less than 0.1 of the maximum document dimension. Finally, $v_0.\textbf{label} == v_1.\textbf{label}$ dictates that v_0 and v_1 have equivalent labels (either they are both "**Header**", "**Question**", or "**Answer**").



(a) Output of DNNs (word-level)

$$
\begin{aligned}
&\textbf{Find}(\\
&\quad \{v_0, v_1\}, \{r_0\},\\
&\quad \textbf{And}(\\
&\quad\quad \textbf{Rel}(v_0, v_1, r_0),\\
&\quad\quad \textbf{And}(r_0\textbf{.label} == \textbf{Right},\\
&\quad\quad\quad \textbf{And}(r_0\textbf{.mag} < 0.1,\\
&\quad\quad\quad\quad v0\textbf{.label} == v1\textbf{.label}))\\
&\quad), \{v_1\}
\end{aligned}
$$

(b) Semantic Entity Grouping Program



(c) Grouped entitties

Figure 2: An example of a program that groups document entities from DNN predictions

If these conditions are met, the program aggregates the valuation of v_1 and this is gathered in the final output.

We specifically save v_0 as the initial entity to be linked with, this means that for every found binding, v_0 is the starting entity in which we want to link every valuation of the V_{set_return} with. For example, consider that we have a binding b where v_0 is assigned with the word "Special", following this aforementioned rule, we find that "Event". If v_0 is instead assigned with "Event", we find v_1's valuation to be "Information". In another case, if v_0 is assigned with "Name" word (belonging to the set "Event Name"), it is expected that the return set of this find program is empty to avoid linking to words that lie outside the semantic entity "Event Name".

6 SYNTHESIZING VRD INFORMATION EXTRACTION PROGRAM

Taking as input the set of documents $\{D\}$ along with the corresponding specifications $\{M\}$, our goal is to synthesize a set of program Ps that is precise (i.e., only links/merges the entities/words that belong to the specs). For this, we first construct a relation set D_R for each document $D \in \{D\}$, as D consists of all entities and D_R consists of all relations between these entities, we can construct a document graph $G = \langle D, D_R \rangle$ for each document. Our synthesizing algorithms take these graph dataset along with the corresponding specifications and is listed in Algorithm 1. Our design of the algorithms is based on 2 key ideas: (1) Iterative goal-oriented refinement of programs and (2) equivalence reduction with term-rewriting.

Iterative goal-oriented refinement of programs Let us take a look at the possible holes of the **Find** function: $\textbf{Find}(\square_1, \square_2, \square_3, \square_4)$, where $\square_i$ is the i^{th} hole, respectively. $\square_1$ denotes the set of variables, $\square_2$ denotes the set of relation variables, $\square_3$ denote the constraint and $\square_4$ denotes the returning set of variables. While enumerating and evaluating all combinations of possible holes is computationally expensive, we can make use of the relation between the variables, and their relations to specify the space of initial programs. The linking between $\square_1$ and $\square_2$ are determined by the *RelC* constraint in $\square_3$. Thus, if we find the effective covering set of relations to cover the specs, we can simultaneously fill both variables and their relation constraints. Thus, instead of enumerating all possible expressions to fill these holes individually, we leverage a mining-based approach to perform initial filling to the find program. While the space of all possible variables and relations is large, ultimately, literature has shown that all it takes is 1 to 2 hops traversal [12, 20] to capture the local relation between entities in semi-structured document information extraction. Inspired by this, our first step is to select the set of possible bindings to be extended upon. Detailed description of this step can be found in Section 6.1 The output of this stage leads to a list of programs represented in the form

$$
\begin{aligned}
&\textbf{Find}(\{v_0, v_1, \ldots v_i\}, \{r_0, r_1, \ldots r_j\},\\
&\quad \textbf{And}(Rel(v_0, r_0, v_1), \textbf{And}(Rel(v_i, r_j, v_k), \ldots)), \{v_i\})
\end{aligned}
$$

This program consists of only variables and relation constraints, which shapes the binding of all variables $v_0, \ldots v_i$ and relation variables $r_0 \ldots r_i$. These bindings are not just initial conditions but also form the constraints within which the program operates. Thus, early pruning of possible bindings leads to a better chance of discovering effective programs. Following this, we proceed and add one constraint at a time toward the goal of discovering high achieving program.

Equivalence reduction with term rewriting Finally, to further prune the possible programs in the enumeration phase, we leverage

Algorithm 1: Overall Program Synthesis

Result: Synthesized programs based on the specified grammar
Input: Graph dataset $\{G\}$, Specs $\{M\}$, Grammar *grammar*
Output: List of synthesized programs PP
$P \leftarrow$ GetInitialPrograms($\{G\}, \{M\}$) (Algorithm 2);
$VS \leftarrow$ CreateVersionSpace(P, $\{G\}$, $\{M\}$);
$PP, CP \leftarrow \{\}, \{\}$;
for $i = 1$ **to** *maxIterations* **do**
  $VS, PP^*, CP^* \leftarrow$ RefinePrograms(VS) (Algorithm 3);
  $PP \leftarrow PP \cup PP^*$;
  $CP \leftarrow CP \cup CP^*$;
end
return PP;

equivalent reduction [21] in the form of term rewriting rules below:

$$\textbf{And}(A, A) \rightsquigarrow A \quad \textbf{Or}(A, A) \rightsquigarrow A$$
$$\textbf{And}(A, \textbf{False}) \rightsquigarrow \textbf{False} \quad \textbf{Or}(A, \textbf{True}) \rightsquigarrow \textbf{True}$$
$$\textbf{And}(A, \textbf{True}) \rightsquigarrow A \quad \textbf{Or}(A, \textbf{False}) \rightsquigarrow A$$
$$F < F \rightsquigarrow \textbf{False} \quad F > F \rightsquigarrow \textbf{False}$$
$$\textbf{Contains}(S, S) \rightsquigarrow \textbf{True}$$
$$\textbf{Equal}(S, S) \rightsquigarrow \textbf{True}$$
$$V.\textbf{label} == V.\textbf{label} \rightsquigarrow \textbf{True}$$
$$R.\textbf{label} == R.\textbf{label} \rightsquigarrow \textbf{True}$$
$$V.\textbf{x0} < V.\textbf{x1} \rightsquigarrow \textbf{True} \quad V.\textbf{x1} < V.\textbf{x0} \rightsquigarrow \textbf{False}$$
$$V.\textbf{y0} < V.\textbf{y1} \rightsquigarrow \textbf{True} \quad V.\textbf{y1} < V.\textbf{y0} \rightsquigarrow \textbf{False}$$

These rules are applied in a bottom-up manner to the program under synthesizing. Below, we give a detailed description of each step of our algorithms.

6.1 Construct initial programs

The goal of initial program construction is to provide a set of well-covered programs. For this stage, we extract the common set of relationships from the document dataset $\{G\}$ and create the initial programs from these paths. More formally, we give the definition of Path in Definition 3. We collect all paths from each linking pair from the specification, followed by constructing the corresponding **Find** programs from these paths. Detailed of these **Find** programs construction are listed in Algorithm 2.

Definition 3. *Let $G = \{D, D_R\}$ be a document graph where D is a symbolic visually rich document defined in Definition 1 and D_R is the relation set between D's entities. A path π from a starting entity s to an ending entity t is a sequence of $\{e_1, e_2, \ldots, e_n\}$, where $e_1 = s$ and $e_n = t$, and each consecutive pair $(e_i, e_{i+1}$ forms an edge in D_R.*

After this step, we obtain the set of initial find programs consisting of only variables, relational variables, and relation constraints between these variables. While these programs can cover a large amount of linking between entities, they are not precise (i.e., they might link wrong entities) since the conditions have yet to be refined. Next, based on these programs, we perform systematic refinement with respect to the specifications.

6.2 Iterative refinement of programs

The ultimate goal of our approach is to synthesize programs that can precisely capture the semantic relationship between entities as stated in Section 3.2 (we want to synthesize programs that do not link wrong entities), we formalize this goal and our detailed refinement process below. Given a graph dataset $\{G\}$ and a specification $\{M\}$ on this graph dataset, for each program p, we collect the set of all possible bindings that satisfy the condition of p as B_p. Among all bindings in B_p, we further divide them into two sets B_p^+ and B_p^-. B_p^+ denotes the set of bindings that *models* the original specifications (i.e., $\forall b \in B_p^+, \forall \mathbf{v}_{ret} \in V_{set\_return}, (b[\mathbf{v}_0], b[\mathbf{v}_{ret}]) \in M$), and B_p^- denotes the set of bindings that do not model the specifications. We can calculate the precision of the program by counting all the linked entities within B_p^+ and B_p^-:

$$prec(B_p^+, B_p^-) = |B_p^+|/(|B_p^+| + |B_p^-|) \tag{18}$$

Algorithm 2: Constructing Initial Find Programs

Result: List of initial find programs
Input: List of document graph $\{G\}$ and corresponding list of specifications $\{M\}$
Output: List of synthesized programs P
Initialize an empty list of programs $P \leftarrow \{\}$;
Initialize an empty list of paths $\Pi \leftarrow \{\}$;
foreach $G_i, M_i \in \{G\}, \{M\}$ **do**
  foreach $e_j, e_k \in M_i$ **do**
    $\Pi \leftarrow \Pi \cup$ GetAllPaths$(G_i, e_j, e_k, 2)$);
  end
end
foreach $\pi \in \Pi$ **do**
  $V_{set}, R_{set}, Rel_{set}, V_{map} \leftarrow \{\}, \{\}, \{\}, \{\}$;
  $rcnt, vcnt, \mathbf{v}_{last} \leftarrow 0, 0, \mathbf{v}_0$;
  foreach $(e_i, e_j) \in \pi$ **do**
    if $e_i \notin V_{map}$ **then**
      $V_{map}[e_i] \leftarrow \mathbf{v}_{vcnt}$;
      $V_{set} \leftarrow V_{set} \cup \{\mathbf{v}_{vcnt}\}, vcnt \leftarrow vcnt + 1$;
    end
    if $e_j \notin V_{map}$ **then**
      $V_{map}[e_j] \leftarrow \mathbf{v}_{vcnt}$;
      $V_{set} \leftarrow V_{set} \cup \{\mathbf{v}_{vcnt}\}, vcnt \leftarrow vcnt + 1$;
    end
    $R_{set} \leftarrow R_{set} \cup \{\mathbf{r}_{rcnt}\}$;
    $Rel_{set} \leftarrow Rel_{set} \cup \{\textbf{Rel}(V_{map}[e_i], \mathbf{r}_{rcnt}, V_{map}[e_j])\}$;
    $rcnt \leftarrow rcnt + 1, \mathbf{v}_{last} \leftarrow V_{map}[e_j]$;
  end
  $RelC \leftarrow$ ConstructRelationConstraint(Rel_{set});
  $P \leftarrow P \cup \{\textbf{Find}(V_{set}, R_{set}, RelC, \{\mathbf{v}_{last}\})\}$;
end
return $\{P\}$;

Our observation is that, given an initial **And** constraint, any additional constraints will reduce either or both B_p^+ and B_p^-. Thus, we keep adding the constraints that improve the *precision* of the program. For every find program p, let $\{C\}$ be the set of constraints that has yet to appear in p and is *valid* towards p,

To keep track of programs performing similarly, we keep version spaces of programs that have similar B_p^+ and B_p^- (i.e., $VS : \langle \mathcal{P}(B), \mathcal{P}(B) \rangle \rightarrow \mathcal{P}(D_P)$), where $\mathcal{P}$ denotes power sets). To avoid synthesizing redundant programs, we the set of covered specifications (i.e., $\bigcup_p B_p^+$), while enumerating, we only accept candidates that cover a part of specifications that have yet to be covered for further extension. This detailed algorithm is listed in Algorithm 3.

At the end of each iteration, we collect additional perfect program set PP^*, the set of perfect negative programs NP^* (programs that only give examples that do not match the specification), and the set of new version space with improved precision. These positive programs and negative programs cover the perfect set *PCover* and *NCover*, respectively. In practice, we also observe the mutually exclusive property between certain entities (See Figure 3): The

Algorithm 3: Refine Programs

Result: Refined Program and Version Spaces
Input: Version spaces VS
Output: List of perfect programs PP^*, perfect negative programs NP^* and version spaces VS^*

$Cover, PCover, NCover = \{\}, \{\}, \{\}$;
$VS^*, PP^*, NP^* \leftarrow \{\}, \{\}, \{\}$;
foreach $B_p^+, B_p^-, p \in VS$ **do**
  if $B_p^+ \setminus Cover = \emptyset$ **then**
    continue;
  end
  foreach *condition* C *based on grammar* **do**
    $B_p^{+*}, B_p^{-*} \leftarrow \text{Filter}(B_p^+, C), \text{Filter}(B_p^-, C)$;
    $p^* \leftarrow \text{AddConstraint}(p, C)$;
    if $B_p^{+*} \setminus PCover \neq \emptyset$ *and* $B_p^{-*} = \emptyset$ **then**
      $PP^* \leftarrow PP^* \cup \{p^*\}$;
      $PCover \leftarrow PCover \cup B_p^{+*}$;
      $Cover \leftarrow Cover \cup B_p^{+*}$;
      continue;
    end
    if $B_p^{+*} \setminus Cover \neq \emptyset$ *and* $prec(B_p^{+*}, B_p^{-*}) > prec(B_p^+, B_p^-)$ **then**
      $Cover \leftarrow Cover \cup B_p^{+*}$;
      $VS^*[B_p^+*, B_p^-*] \leftarrow VS^*[B_p^+*, B_p^-*] \cup \{p^*\}$;
    end
    if $B_p^{+*} = \emptyset$ *and* $B_p^{-*} \setminus NCover \neq \emptyset$ **then**
      $NP^* \leftarrow NP^* \cup \{p^*\}$;
      $NCover \leftarrow NCover \cup B_p^{-*}$;
    end
  end
end
$EPCover \leftarrow \{\}$ $p_u = \textbf{Union}(\{PP*\} \cup \{NP*\})$;
foreach $B_p^+, B_p^-, p \in VS$ **do**
  if $B_p^- \setminus (PCover \cup NCover)$ *and* $B_p^+ \setminus PCover \neq \emptyset$ **then**
    $PP^* \leftarrow PP^* \cup \textbf{Exclude}(p, p_u)$;
    $EPCover \leftarrow EPCover \cup B_p^+$;
  end
end
$PCover \leftarrow EPCover \cup PCover$;
foreach $B_p^+, B_p^-, p \in VS^*$ **do**
  if $B_p^+ \setminus PCover = \emptyset$ **then**
    $VS^* \leftarrow VS^* \setminus \{(B_p^+, B_p^-, p)\}$;
  end
end

exclusive rule for name in this case can be specified as:

$$\begin{aligned}
&\textbf{Exclude}(\\
&\quad \textbf{Find}(\{v_0, v_1\}, \{r_0\},\\
&\qquad \textbf{And}(r_0.\textbf{lbl} == L_0, \textbf{Contains}(v_0.\textbf{text}, \text{" : "})),\\
&\qquad \{v_1\}),\\
&\quad \textbf{Find}(\{v_0, v_1\}, \{r_0\},\\
&\qquad \textbf{And}(r_0.\textbf{lbl} == L_0, \textbf{Contains}(v_0.\textbf{text}, \text{" – "})),\\
&\qquad \{v_1\})\\
&)
\end{aligned}$$

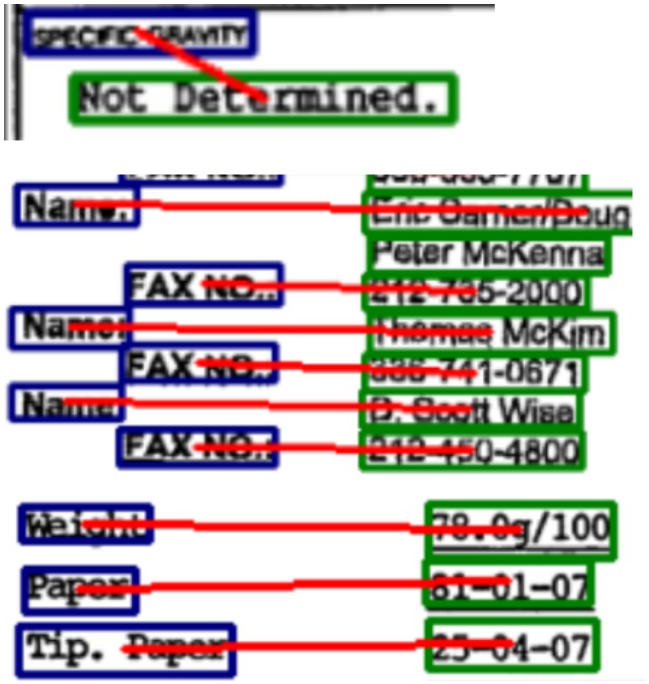


Figure 3: Mutually-exclusive between semantic relations: while normally, key can be linked towards values on the down right, ("specific gravity" towards "not determined"), in the case of Name and Fax Number, since the horizontal linking rules are already been applied, downright linking should not be applied anymore: Linking between one name to its value should exclude other fax's values

Meaning that we would take the value on the right, but do not take the values that already belong to the other rule (containing "-").

Finally, to construct the final program, we perform union on all of these find programs and exclude the union of all negative programs:

$$p_{final} := \textbf{Exclude}(\textbf{Union}(pp \in PP), \textbf{Union}np \in NP))$$

7 EXPERIMENT

We implement VRDSynth in Python with over 7,000 lines of code. The experiments are performed on a computer equipped with AMD Ryzen Threadripper PRO 3995WX, with 128 GB of RAM and with RTX 4090 GPU with 24GB of VRAM. We set the maximum synthesis time of VRDSynth to be 2 hours on all tasks, since the programs can be synthesized offline, this can be seen as analogous to training time for machine learning models.

Benchmark We evaluate VRDSynth on semantic entity linking task on the FUNSD [13] and XFUND [22] datasets. These two dataset consists of collections of 200 forms in 8 languages: English (en), German (de), French (fr), Spanish (es), Italian (it), Japanese (ja), Portuguese (pt), and Chinese (zh). For each language, this benchmark consists of 200 scanned and annotated forms from various domains. We use the standard division of 150 forms to perform synthesis and 50 forms to test the synthesized programs.

Baselines We compare VRDSynth with the publicly released replication package along with the pre-trained model of LayoutXLM [14] in , the state-of-the-art pre-trained multi-language model.

7.1 RQ1: Effectiveness of VRDSynth

We follow the single-task settings [22]. For VRDSynth, we use the single DSL for all languages. For each specific language, we use the training documents of the language to synthesize programs and evaluate the synthesized programs collectively on the test set. For LayoutXLM [14], we use the training documents for fine-tuning and evaluating the trained model on the language's test set. We record precision, recall, and F1 score for both VRDSynth's synthesized program and LayoutXLM [14] in Table 1.

Regarding F1 score, VRDSynth outperforms LayoutXLM in 6 out of 8 languages. Notably, in English and Portuguese, VRDSynth achieved 35% and 22.9% higher F1 scores than LayoutXLM, respectively. These better F1 scores are largely due to the contribution of the better precisions: on average, VRDSynth's precision (0.693) is 61.5% higher than LayoutXLM (0.429). In terms of recall, LayoutXLM is generally better than VRDSynth (with overall 51.8% higher recall), with an exception on English semantic entity linking, where VRDSynth achieved both higher precision and recall. While VRDSynth and LayoutXLM's both used the same training set for synthesizing and fine-tuning and VRDSynth's recall can be improved, VRDSynth achieved better precisions and F1 scores without requiring any pre-training data hints the better capability in generalizing to additional languages.

RQ1 Conclusion: VRDSynth achieved average precision, recall, and F1 score of 0.693, 0.465, and 0.551 respectively without requiring pre-trained data or pre-training procedure and outperformed LayoutXLM on 6 out of 8 languages.

7.2 RQ2: Contribution of joining negative and mutually-exclusive programs

To measure the contribution of joining programs, we perform a comparison of synthesized programs' performance before and after adding counter programs in terms of F1 score in Figure 4.

The usage of joining consistently improves the F1 score of VRDSynth in all languages. In detail, using filtering gives improvements of 10.1%, 15.2%, 17.5%, 1.5%, 10.9%, 21.4%, 15.2%, 5.1% on en, de, es, fr, it, ja, pt and zh respectively. This constitutes an overall improvement of 12.1% in terms of F1 score.

RQ2 conclusion: Joining programs give an overall improvement of 12.1% in F1 score. Out of 8 languages, the improvements in 6 languages (en, de, es, it, ja, zh) are over 10%.

7.3 RQ3: Efficiency of VRDSynth

We measure the synthesizing and inference time of VRDSynth's variants - VRDSynth$_{prec}$ and VRDSynth$_{full}$ (without and with joining programs) along with LayoutXLM. To provide a precise comparison, we set the number of available CPU cores to 1 and do not use GPU.

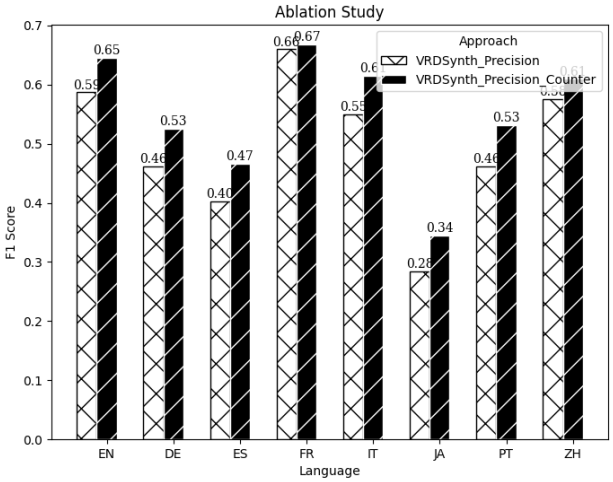


Figure 4: Comparison between VRDSynth with and without the usage of joining programs.

7.3.1 RQ3.1. Inference time efficiency. We note down the mean and standard deviation of runtimes using the synthesized programs using different VRDSynth's variants and LayoutXLM in Table 2. VRDSynth inference time is 4.15 seconds on average, while it is slightly less efficient than LayoutXLM, the speed is acceptable. The non-joining version of VRDSynth is more efficient, running on average of only 1 second per document. We note that VRDSynth's inference is implemented in Python, while LayoutXLM is based on Pytorch, which is implemented in C++, thus, in future implementation, it should be interesting to see the improvement in using C++ for executing VRDSynth's synthesized programs.

7.3.2 RQ3.2. Memory efficiency. We also measure the storage and maximum memory usage of VRDSynth and LayoutXLM in Table 3.

RQ3 Conclusion: Overall VRDSynth has an average inference time of 4.15 seconds per document, which is comparable with LayoutXLM despite being implemented in Python instead of C++. For memory efficiency, VRDSynth requires only 1 MB of storage and 380 MB of memory for inference, significantly lower than LayoutXLM at 1.48GB for storage and 3.04GB for memory inference.

8 CONCLUSION AND FUTURE WORKS

We introduced a synthesis-based approach towards visually rich document understanding. In detail, we proposed a new DSL allowing the description of programs that link entities based on spatial and textual constraints. We further made use of this DSL and proposed a novel synthesis algorithm, VRDSynth that efficiently synthesizes programs based on the DSL. The synthesized programs outperformed LayoutXLM, the state-of-the-art in multi-language document understanding in 6 out of 8 languages in the joint benchmark of FUNSD and XFUND. Our evaluation further shows that the technique of joining and mutually excluding negative programs

Table 1: Comparison of Performance between VRDSynth and LayoutXLM, Prec., Rec. are abbreviations for precision and recall. EN, DE, ES, FR, IT, JA, PT, and ZH stand for English, German, Spanish, French, Italian, Japanese, Portuguese and Chinese respectively.

Method	Metrics	EN	DE	ES	FR	IT	JA	PT	ZH	Avg.
VRDSynth (Full)	Prec.	**0.705**	**0.7**	**0.617**	**0.74**	**0.71**	**0.586**	**0.706**	**0.781**	**0.693**
	Rec.	**0.594**	0.422	0.374	0.609	0.543	0.245	0.426	0.495	0.465
	F1	**0.645**	**0.526**	0.466	**0.668**	**0.615**	0.345	**0.531**	**0.61**	**0.551**
LayoutXLM (Base)	Prec.	0.415	0.426	0.482	0.423	0.458	0.475	0.32	0.439	0.429
	Rec.	0.516	**0.683**	**0.769**	**0.751**	**0.706**	**0.801**	**0.664**	**0.756**	**0.706**
	F1	0.478	**0.524**	**0.592**	0.541	0.555	**0.597**	0.432	0.558	0.535

Table 2: Inference time comparison of VRDSynth's variants and LayoutXLM, all measurements are in seconds

Lang	VRDSynth$_{full}$		VRDSynth$_{prec}$		LayoutXLM	
	Mean	**Std.**	**Mean**	**Std.**	**Mean**	**Std.**
En	1.68	1.27	0.64	0.495	1.88	0.41
De	3.71	2.96	0.86	0.69	2.45	0.85
Es	5.23	2.57	1.27	0.63	2.80	1.02
Fr	3.91	3.45	0.80	0.56	2.68	1.17
It	4.99	3.30	1.12	0.75	3.42	1.33
Ja	2.87	2.71	1.07	1.02	4.99	2.46
Pt	5.76	6.23	1.34	1.42	3.20	1.19
Zh	5.06	2.434	1.13	0.54	4.35	1.99
Avg.	4.15	2.95	1.03	0.76	3.22	1.3

Table 3: Storage and memory efficiency of VRDSynth's variants and LayoutXLM

	VRDSynth$_{full}$	VRDSynth$_{prec}$	LayoutXLM
Storage	1020 KB	424 KB	1.48 GB
Memory	380 MB	378 MB	3.04 GB

gives a boost in terms of synthesized program effectiveness and that they are memory and computationally efficient in using.

REFERENCES

[1] Arun Iyer, Manohar Jonnalagedda, Suresh Parthasarathy, and Sriram Rajamani. Synthesis and Machine Learning for Heterogeneous Extraction.

[2] Yiheng Xu, Minghao Li, Lei Cui, Shaohan Huang, Furu Wei, and Ming Zhou. Layoutlm: Pre-training of text and layout for document image understanding, 2019.

[3] Yang Xu, Yiheng Xu, Tengchao Lv, Lei Cui, Furu Wei, Guoxin Wang, Yijuan Lu, Dinei Florencio, Cha Zhang, Wanxiang Che, Min Zhang, and Lidong Zhou. LayoutLMv2: Multi-modal pre-training for visually-rich document understanding. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 2579–2591, Online, August 2021. Association for Computational Linguistics.

[4] Yupan Huang, Tengchao Lv, Lei Cui, Yutong Lu, and Furu Wei. Layoutlmv3: Pre-training for document ai with unified text and image masking. In *Proceedings of the 30th ACM International Conference on Multimedia*, 2022.

[5] Xiaojing Liu, Feiyu Gao, Qiong Zhang, and Huasha Zhao. Graph Convolution for Multimodal Information Extraction from Visually Rich Documents. 2019.

[6] D. Lohani, A. Belaïd, and Y. Belaïd. An Invoice Reading System Using a Graph Convolutional Network. pages 144–158. 2019.

[7] Vishal Sunder, Ashwin Srinivasan, Lovekesh Vig, Gautam Shroff, and Rohit Rahul. One-shot Information Extraction from Document Images using Neuro-Deductive Program Synthesis. jun 2019.

[8] Nguyen Dang Tuan Anh and Nguyen Thanh Dat. End-to-End Information Extraction by Character-Level Embedding and Multi-Stage. *British Machine Vision Conference (BMVC)*, pages 1–12, 2019.

[9] Anoop Raveendra Katti, Christian Reisswig, Cordula Guder, Sebastian Brarda, Steffen Bickel, Johannes Höhne, and Jean Baptiste Faddoul. Chargrid: Towards Understanding 2D Documents. 2018.

[10] Rasmus Berg Palm, Ole Winther, and Florian Laws. CloudScan - A Configuration-Free Invoice Analysis System Using Recurrent Neural Networks. In *Proceedings of the International Conference on Document Analysis and Recognition, ICDAR*, 2018.

[11] Hanchuan Peng, Fuhui Long, Wan-Chi Siu, Zheru Chi, and David Dagan Feng. Document image matching based on component blocks. In *Proceedings 2000 International Conference on Image Processing (Cat. No.00CH37101)*, volume 22, pages 601–604 vol.2. IEEE, 2000.

[12] Hanchuan Peng, Fuhui Long, and Zheru Chi. Document image recognition based on template matching of component block projections. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 25(9):1188–1192, sep 2003.

[13] Guillaume Jaume, Hazim Kemal Ekenel, and Jean-Philippe Thiran. FUNSD: A Dataset for Form Understanding in Noisy Scanned Documents. may 2019.

[14] Yiheng Xu, Tengchao Lv, Lei Cui, Guoxin Wang, Yijuan Lu, Dinei Florencio, Cha Zhang, and Furu Wei. Layoutxlm: Multimodal pre-training for multilingual visually-rich document understanding. 2021.

[15] Niels Rogge. Fine-tune layoutxlm on xfund (relation extraction). https://github.com/NielsRogge/Transformers-Tutorials/blob/master/LayoutXLM/Fine_tune_LayoutXLM_on_XFUND_(relation_extraction).ipynb, 2023. Accessed: 2023-12.

[16] Sumit Gulwani, Oleksandr Polozov, Rishabh Singh, and Boston Delft. *Program Synthesis*. 2017.

[17] Sumit Gulwani. Automating string processing in spreadsheets using input-output examples. *SIGPLAN Not.*, 46(1):317–330, jan 2011.

[18] Maxwell Nye, Luke Hewitt, Joshua Tenenbaum, and Armando Solar-Lezama. Learning to Infer Program Sketches. feb 2019.

[19] José Cambronero, Sumit Gulwani, Vu Le, Daniel Perelman, Arjun Radhakrishna, Clint Simon, and Ashish Tiwari. Flashfill++: Scaling programming by example by cutting to the chase. *Proceedings of the ACM on Programming Languages*, 7(POPL):952–981, January 2023.

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

[21] Calvin Smith and Aws Albarghouthi. Program synthesis with equivalence reduction. In Constantin Enea and Ruzica Piskac, editors, *Verification, Model Checking, and Abstract Interpretation*, pages 24–47, Cham, 2019. Springer International Publishing.

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

[23] Syed Saqib Bukhari, Faisal Shafait, and Thomas Breuel. Coupled Snakelet Model for Curled Textline Segmentation of Camera-Captured Document Images. *Proceedings of the 10th International Conference on Document Analysis and Recognition. International Conference on Document Analysis and Recognition (ICDAR-09), July 26-29, Barcelona, Spain*, pages 61–65, 2009.

[24] Syed Saqib Bukhari, Faisal Shafait, and Thomas M Breuel. High Performance Layout Analysis of Arabic and Urdu Document Images. *Icdar*, pages 1275–1279, 2011.

[25] Byeongyong Ahn, Jewoong Ryu, Hyung Il Koo, and Nam Ik Cho. Textline detection in degraded historical document images. *Eurasip Journal on Image and Video Processing*, 2017(1), 2017.
[26] Dr Ciravegna et al. Adaptive information extraction from text by rule induction and generalisation. 2001.
[27] Li Yujian and Liu Bo. A normalized levenshtein distance metric. *IEEE transactions on pattern analysis and machine intelligence*, 29(6):1091–1095, 2007.
[28] Marcal Rusinol, Tayeb Benkhelfallah, and Vincent Poulain Dandecy. Field extraction from administrative documents by incremental structural templates. *Proceedings of the International Conference on Document Analysis and Recognition, ICDAR*, pages 1100–1104, 2013.
[29] Ondrej Chum, James Philbin, Andrew Zisserman, et al. Near duplicate image detection: min-hash and tf-idf weighting. In *BMVC*, volume 810, pages 812–815, 2008.
[30] Vincent Poulain d'Andecy, Emmanuel Hartmann, and Marçal Rusinol. Field extraction by hybrid incremental and a-priori structural templates. In *2018 13th IAPR International Workshop on Document Analysis Systems (DAS)*, pages 251–256. IEEE, 2018.
[31] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.
[32] Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *Advances in neural information processing systems*, pages 1024–1034, 2017.
[33] Haoyuan Peng, Lu Liu, Yi Zhou, Junying Zhou, and Xiaoqing Zheng. Attention-based Belief or Disbelief Feature Extraction for Dependency Parsing. *AAAI*, (Zheng), 2018.
[34] Linfeng Song, Yue Zhang, Zhiguo Wang, and Daniel Gildea. N-ary relation extraction using graph state lstm. *arXiv preprint arXiv:1808.09101*, 2018.
[35] Shah Rukh Qasim, Hassan Mahmood, and Faisal Shafait. Rethinking Table Parsing using Graph Neural Networks. may 2019.
[36] Shah Rukh Qasim, Hassan Mahmood, and Faisal Shafait. Rethinking Table Recognition using Graph Neural Networks. may 2019.
[37] Wenlin Wang, Zhe Gan, Hongteng Xu, Ruiyi Zhang, Guoyin Wang, Dinghan Shen, Changyou Chen, and Lawrence Carin. Topic-Guided Variational Autoencoders for Text Generation. mar 2019.
[38] Armando Solar-Lezama. *Program Synthesis by Sketching*. PhD thesis, USA, 2008. AAI3353225.
[39] Mohammad Raza and Sumit Gulwani. Web data extraction using hybrid program synthesis: A combination of top-down and bottom-up inference. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, SIGMOD '20, page 1967–1978, New York, NY, USA, 2020. Association for Computing Machinery.
[40] Celeste Barnaby, Qiaochu Chen, Roopsha Samanta, and Işıl Dillig. Imageeye: Batch image processing using program synthesis. *Proc. ACM Program. Lang.*, 7(PLDI), jun 2023.