International Journal on Document Analysis and Recognition (IJDAR) (2024) 27:395–414
https://doi.org/10.1007/s10032-024-00486-7

SPECIAL ISSUE PAPER

ChemScraper: leveraging PDF graphics instructions for molecular diagram parsing

Ayush Kumar Shah[1] · Bryan Amador[1] · Abhisek Dey[1] · Ming Creekmore[1] · Blake Ocampo[2] · Scott Denmark[2] · Richard Zanibbi[1]

Received: 15 November 2023 / Revised: 31 May 2024 / Accepted: 5 June 2024 / Published online: 5 July 2024

Abstract

Most molecular diagram parsers recover chemical structure from raster images (e.g., PNGs). However, many PDFs include commands giving explicit locations and shapes for characters, lines, and polygons. We present a new parser that uses these born-digital PDF primitives as input. The parsing model is fast and accurate, and does not require GPUs, Optical Character Recognition (OCR), or vectorization. We use the parser to annotate raster images and then train a new multi-task neural network for recognizing molecules in raster images. We evaluate our parsers using SMILES and standard benchmarks, along with a novel evaluation protocol comparing molecular graphs directly that supports automatic error compilation and reveals errors missed by SMILES-based evaluation. On the synthetic USPTO benchmark, our born-digital parser obtains a recognition rate of 98.4% (1% higher than previous models) and our relatively simple neural parser for raster images obtains a rate of 85% using less training data than existing neural approaches (thousands vs. millions of molecules).

Keywords Graphics recognition · Data generation · Evaluation · PDF · Chemoinformatics

1 Introduction

We address a pressing need for robust systems to extract molecule drawings from PDF files. Such systems facilitate data mining applications for chemoinformatics, multi-modal chemical search, and chemical reaction planning.

Current molecule structure recognizers generally parse images from pixel-based raster images, and produce chemical structure descriptions such as Simplified Molecular-Input Line-Entry System strings (SMILES [45]) as output. A number of these approaches work well, and some include modern variations of encoder/decoder models that recognize structure with high accuracy (see Sect. 2).

However, modern documents often use vector images to depict molecules. Vector images encode diagrams as characters, lines, and other graphic primitives. We wish to use PDF drawing instructions directly to produce fast, accurate methods for indexing molecule images. We were motivated to use PDF instructions by earlier math formula recognition work by Baker et al. using a combination of PDF instructions and image analysis [3]. In our approach, only PDF instructions are used. In Sect. 4 we describe our improved `SymbolScraper` tool [38] that extracts PDF instructions without image processing.

Ayush Kumar Shah and Bryan Amador have contributed equally to this work.

✉ Ayush Kumar Shah
as1211@rit.edu

Bryan Amador
ma5339@rit.edu

Abhisek Dey
ad4529@rit.edu

Ming Creekmore
mec5765@rit.edu

Blake Ocampo
blakeo2@illinois.edu

Scott Denmark
sdenmark@illinois.edu

Richard Zanibbi
rxzvcs@rit.edu

[1] Document and Pattern Recognition Lab, Rochester Institute of Technology, Rochester, NY, USA

[2] Department of Chemistry, University of Illinois at Urbana-Champaign, Champaign, IL, USA

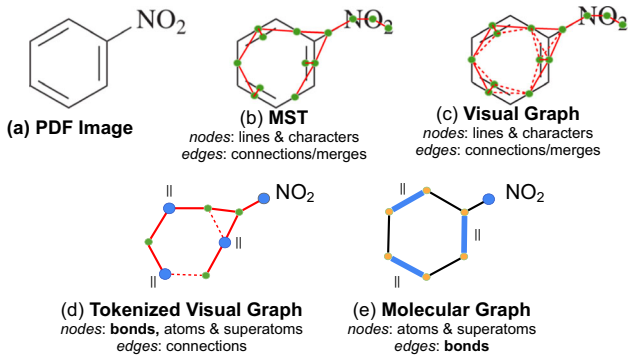


Fig. 1 Parsing nitrobenzene ($C_6H_5NO_2$) from a PDF image (**a**). **b** *Minimum Spanning Tree (MST)* over lines & characters. (**c**) *Visual Graph* with additional edges (dashed lines). (**d**) *Tokenized Visual Graph* with merged nodes (bonds and named groups). (**e**) *Molecular Graph*. Blue nodes show double bonds and atom/group names in (**d**, **e**). In **e** orange nodes are 'hidden' carbon atoms, and single/double bonds are converted from nodes to edges

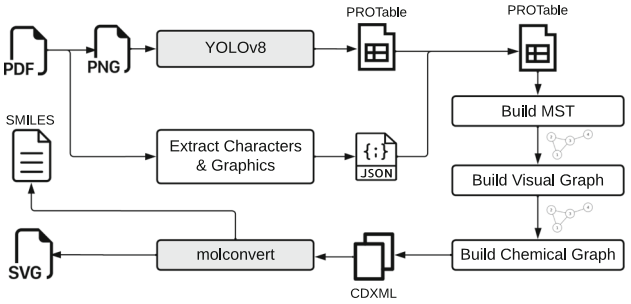


Fig. 2 ChemScraper born-digital pipeline. Molecules are detected in PNG page images, but symbols are extracted from PDF instructions. Page-Region-Object tables store bounding boxes and the graphics they contain. Molecules are recognized in three stages, producing CDXML containing the page location, appearance, and chemical structure for each. CDXML can then be converted to chemical structure file formats (e.g., SMILES) or rendered as images (e.g., SVG)

In Sect. 4 we describe the ChemScraper born-digital parser, which is fast and simple in design.[1] As illustrated in Fig. 1, starting from PDF graphical primitives, first a Minimum Spanning Tree (MST) is constructed to identify neighboring primitives. Additional edges between primitives are added, and edges to floating objects removed to capture the *visual* structure of the diagram. Primitives are then grouped (i.e., *tokenized*) into molecular entities including atom/superatom names and bonds. Finally, graph transformations convert the tokenized visual graph into a graph representing molecular structure.

This *born-digital* vector image parser is one component in the online ChemScraper molecule extraction tool,[2] which includes a YOLOv8 [43] detection module not described in this paper. Figure 2 provides an overview of the full ChemScraper born-digital extraction pipeline. The model locates page regions where molecular diagrams appear, and then parses their structure. Recognized molecules are stored in ChemDraw[3] CDXML files [26]. CDXML represents both visual and chemical structure in molecular diagrams. The ChemAxon `molconvert` command line tool[4] is used to convert CDXML to vector images (SVG) and SMILES. Recognized molecules can then be used for editing, search, and other applications (e.g., in chemoinformatics).

We also use the born-digital parser to annotate pixel-based raster images, to address a shortage of such data. This includes annotations for all graphical primitives, atoms, and bonds (see Sect. 5). We use this data to train a new *visual* parser, a novel multi-task neural network for recognizing molecule diagrams in raster images (see Sect. 6). The visual parser starts by creating line-shaped contour primitives from a raster image that over-segment lines and characters. Just as for the born-digital parser, the visual parser creates a visual graph providing an explicit correspondence between an input image and recognized structure, after which the same tokenization and molecular graph generation steps used for the born-digital parser are performed. In contrast to recent approaches the neural network is *segmentation-aware,* and in recurrent runs, input features associated with primitives are updated.

In Sect. 7, we evaluate our born-digital and visual parsers with two representations: SMILES and labeled directed graphs. Direct comparison of molecular structure graphs in evaluation is a contribution of this paper: it supports automatic compilation of structural differences. In addition, we report structural differences that are missed in SMILES-based evaluation.

In the next section, we summarize prior work in chemical structure recognition.

2 Related work

We begin by surveying approaches to parsing molecular structure, categorizing them into (1) rule-based systems, and (2) neural-based systems. For neural-based systems, we further divide these into methods that produce string representations of structure (e.g., SELFIES [18], DeepSMILES [27], or InChI [13, 14]) and methods that produce graph representations of structure.

While our focus is parsing molecular diagrams, we wish to briefly acknowledge recent work in detecting diagrams. This includes using YOLOv8, an updated version of Scaled

[1] Publicly available code/tools: https://gitlab.com/dprl/graphics-extraction/-/tree/icdar2024.

[2] https://chemscraper.frontend.staging.mmli1.ncsa.illinois.edu/configuration.

[3] https://revvitysignals.com/products/research/chemdraw.

[4] https://docs.chemaxon.com/display/docs/molconvert_index.md.

YOLOv4 [43] with performance and efficiency enhancements. In earlier work, Sun et al. [41] use a convolutional network, addressing scale issues using Spatial Pyramidal Pooling (SPP) [11]. Their approach outperformed popular detection models of the time, including Faster R-CNN and SSD.

2.1 Rule-based parsers

The earliest parser for chemical diagrams in printed documents we know of is a rule-based parser by Ray et al. from the late 1950s [33]. This approach first detected atoms in scanned document images, and then connections between atoms were identified in the regions between atoms. Rules based on the number of connections for atoms were used to determine the type of bonds, which worked well for common compounds.

An important later development was the creation of the Kekulé system [22]. Kekulé adds additional pre-processing and improved visual detection of bond types over previous methods. Kekulé used thinning and vectorization of raster scans to eliminate variations in bond lines and characters, and ensured that a consistent set of characters and lines were recovered. Once a connection between a pair of atoms was established, the system visually detected the bond type instead of using chemical rules as Ray et al. did. In the same period, CLiDE [17] added the use of connected component analysis in disconnected bond groups to identify bond types. The final adjacency matrix for structure was created similar to Kekulé. Another system by Comelli et al. [6] used additional processing to identify charges as subscripts or superscripts attached to atoms.

A still-popular open-source system extending the rules of CLiDE and Kekulé is OSRA by Filipov et al. [9]. OSRA refined processing of raster images generated from born-digital documents, which tend to have clearly rendered text lines, characters, and graphics. A similar system is MolRec [36], which uses horizontal and vertical grouping to detect connected atoms, their charge, and stereochemical information. The more recent CSR system [4] also uses rule-based graphical processing to output SMILES representations for molecules, using the *OpenBabel* [28] toolkit to generate a valid connectivity table.

2.2 Neural networks

String Output. Recent advances in neural networks have proven effective for parsing chemical diagrams. For example, Staker et al. [40] use an end-to-end model for extracting molecular diagrams from documents and converting them into SMILES strings. For diagram extraction, they used a U-Net [34] to segment diagrams, which were then passed through an attention-based encoder network [42] to generate a SMILES string representing molecular structure from the segmented image.

DECIMER [32] also uses an encoder–decoder model for extracting molecular structure from raster images. In their work they explored using different structure representations, including SMILES, DeepSMILES, and SELFIES. They found that SELFIES produced stronger results because of the additional information encoded in comparison with SMILES strings.

Additional encoder–decoder parsers include IMG2SMI by Campos et al. [5] which uses a Resnet-101 [12] backbone to extract image features. Li et al. [19] modified a TNT vision transformer encoder [10] by adding an additional decoder. This use of a vision transformer was made possible by the BMS (Bristol–Myers–Squibb) dataset [2] released by Kaggle, which provided a larger baseline for the conversion of molecule images to InChI (International Chemical Identifier names). The training dataset used by Li et al. contained 4 million molecule images. Similarly, SwinOCSR by Xu et al. [47] used the Swin transformer to encode image features and another transformer-based decoder to generate DeepSMILES, and used a focal loss to address the token imbalance problem in text representations of molecular diagrams.

Graph Output. String representations of molecular structure lack direct geometric representation between input objects (e.g., atoms and bonds) and the output strings, and models trained upon them require extensive training data [23]. In recent years, molecular diagram parsers that combine rule-based and neural-based approaches and generate graph representations have emerged. These methods usually employ a graph decoder or graph construction algorithm.

MolScribe [30] uses a SWIN transformer to encode molecular images and a graph decoder consisting of a 6-layer transformer to jointly predict atoms, bonds, and layouts, yielding a 2D molecular graph structure. They also incorporate rule-based constraints for chirality (i.e., 3D topology) and algorithms to expand abbreviations.

MolGrapher [23] is another method employing a graph-based output representation. It utilizes a ResNet-18 backbone to locate atoms, and constructs a supergraph incorporating all feasible atoms and bonds as nodes, which is then constrained. Subsequently, a Graph Neural Network (GNN) is applied to the supergraph, accompanied by external Optical Character Recognition (OCR) for node classification. Both these systems utilize multiple data augmentation strategies, including diverse rendering parameters, such as font, bond width, bond length, and random transformations of atom groups, bonds, abbreviations, and R-groups (i.e., abbreviations for 'rest of molecule') to bolster model robustness.

Likewise, Yoo et al. [48] and OCMR [44] produce graph-based outputs directly from molecular images. Yoo et al.

[48] leverage a ResNet-34 backbone, followed by a Transformer encoder equipped with auxiliary atom number and label classifiers. A transformer graph decoder with self-attention mechanisms is used for bonds. In contrast, Wang et al. [44] employ multiple neural network models for different parsing steps. These steps include key-point detection, character detection, abbreviation recognition, atomic group reconstruction, atom and bond prediction. A graph construction algorithm is subsequently applied to the outputs.

These graph-based methods offer improved interpretability and robustness, and represent chemical structures naturally. In particular, atom-level alignment with input images facilitates easy examination, geometric reasoning, and correction of predicted results.

3 ChemScraper parsers

In this paper we present two parsers: one parses molecule diagrams in PDF directly from PDF drawing instructions (vector images), while the other recognizes molecules from raster images (pixel-based). Both parsers use a compiler-style multi-step architecture that (1) identifies input primitives, (2) recovers *visible* diagram structure, and then (3) converts visible structure to chemical structure information.

The born-digital parsers' use of Minimum Spanning Trees (MSTs) to recognize molecular diagrams is novel. The detailed PDF graphics information recovered by SymbolScraper is also novel: both as a new data source, and in its application to fast and accurate structure recognition.

To simplify the recognition task, our visual parser operates bottom-up from image region primitives that oversegment lines and characters. The parser is a multi-task, segmentation-aware neural network. The network is run repeatedly until the segmentation (i.e., merging) of primitives remains unchanged. Unlike most recent models, the learning framework utilizes *explicit* segmentation hypotheses, in contrast to 'segmentation-free' models generating descriptions of structure without image region correspondences. To support recurrent execution of the network as segmentation changes, we also introduce a novel discrete attention mechanism: images used for classifier input are generated from primitive contours, and are dynamically updated as larger candidate symbols and associated neighborhoods are identified. Similar to other models described above, a ResNet-based convolutional backbone is used for features. However, images of the same size are used for both query and context images, and they are passed separately through the backbone.

Like previous methods, chemical constraints are used to increase accuracy and simplify parser design. Both parsers produce the same visual structure graph representation as illustrated in Fig. 1c, and then use the same subsequent steps to tokenize names/bonds and then identify chemical structure. The regular structure of molecular diagrams motivates using simple visual features, and taking a divide-and-conquer approach to recovering structure. Structure is recovered based on neighboring MST primitives for the born-digital parser, and from small overlapping neighborhoods (windows) in the visual parser.

An important attribute of ChemScraper output graphs is that they contain both visual and chemical structure information. This allows output graphs to closely match their original appearance in addition to capturing chemical structure. The additional visual information is helpful both for reusing the appearance of molecules within documents, and for visualization and checking of recognition results.

4 Born-digital parser

In this section we present the ChemScraper born-digital parser for recognizing molecular diagrams directly from vectorized PDF images. As seen in Fig. 3, our born-digital parser has four stages, including extracting graphics commands using an improved SymbolScraper [38], constructing a Minimum Spanning Tree (MST), rewriting the MST as a visual structure graph, and finally rewriting the visual graph into a molecular structure graph. The final molecular graph replaces line intersections by carbon atoms, and all bond tokens/nodes (e.g., single, double, triple, solid/hashed wedge) are replaced by edges.

This is a compiler-like recognition architecture, with some similarities to the DRACULAE mathematical formula recognition system [49]. Using a compiler-based architecture provides a helpful separation of concerns that allows changes to be implemented and tested across smaller modules.

We provide an overview of the outputs and processing for stages shown in Fig. 3. Each stage is then described in more detail in the remainder of this section. The full parsing process has an asymptotic run-time complexity of $O(n^2 \log n)$ for n nodes in the input graph (PDF character/graphics primitives), reflecting the cost of MST construction.

Stages 1 & 2: Primitive Graph (MST). SymbolScraper recovers primitive symbols from PDF, for which neighboring objects are identified using an MST. Because molecule diagrams represent connections between atoms/groups using line intersections and line/character proximity, MSTs capture many valid connections. However MSTs prune cycles, some primitives must be merged, and some diagrams contain multiple molecules (e.g., parallel lines in bonds and floating ions).

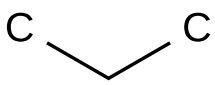
Stage 3: (Tokenized) visual graph. To capture structure missing in the primitive MST, the MST is transformed to provide a two-dimensional syntactic analysis for the visible

Input: Born-Digital PDF Molecule Image

1. **Extract Symbols from PDF**
   Characters and graphical objects (e.g., lines)
2. **Build Minimum Spanning Tree (MST)**
   Connect neighboring lines, shapes, & characters
3. **MST → Visual Graph**
   (a) Detect negative charges (vs. other lines)
   (b) Restructure MST
   (+) add edges: touching lines (e.g., in rings), adjacent parallel lines and char/line pairs
   (-) delete edges: 'floating' objects
   (c) **Tokenization**
   · Neighboring characters → name nodes
   · Neighboring parallel lines → bond nodes
4. **Visual Graph → Molecular Graph**
   *No tunable parameters
   (a) Convert line intersections into carbons
   (b) Replace bond nodes by edges
   (c) Annotate names with subgraphs (e.g., SO_2)
   (d) Generate CDXML

Output: Editable molecular diagram (CDXML)

Fig. 3 Molecule parsing from PDF symbols. Symbol information is transformed into an MST (Fig. 1b), a visual structure graph (Fig. 1c), a *tokenized* visual graph (Fig. 1d), finally a molecular structure graph (Fig. 1e)



(a) Born-Digital PDF for Propane (C_3H_8). Note non-visible (implicit) carbon and hydrogen atoms

```
1 0 0 -1 0 75 cm
45.926 36.102 m
106.832 71.266 l
```

(b) Instructions for Leftmost Line in PDF Image

```
{
    "typeFromPDF": "line",
    "graphicObjectID": 0,
    "length": 70.32814383876341,
    "angle": 330.00006986692745,
    "lineWidth": 3.333334,
    "points": [
        {"x": 44.48262170992254,
         "y": 39.73133054974975},
        {"x": 108.27537771024348,
         "y": 2.9006694197326697}
    ]
}
```

(c) `SymbolScraper` JSON for Leftmost Line in (a)

Fig. 4 Extracting symbols from PDF image (**a**). **b** *cm* is a *context matrix* defining an affine transformation for subsequent objects. *m* moves the cursor to a point, and *l* draws a line from the cursor to the specified point. **c** Line endpoints, angle, and width are extracted by SymbolScraper

primitives. This is done by first adding/removing edges to correct MST structure producing a *visual structure graph* (Fig. 1c), followed by grouping characters and lines into names and bond types (i.e., tokens) producing a *tokenized visual structure graph* (Fig. 1d).

Stage 4: Molecular Graph. The final stage is semantic analysis: visual syntax is mapped to represented information/structure, including elements not visible in the diagram. This includes identifying hidden carbon atoms at line intersections, and structures represented only by name. In our system, names are mapped to molecular subgraphs using a dictionary. In Fig. 1e, NO_2 will be replaced by a subgraph with one nitrogen and two oxygen atoms connected to a hidden carbon.

The semantic analyzer can also be reused with any parser producing visual graphs in the expected format, and we use this with the visual parser presented later in Sect. 6.

4.1 Extracting symbols from PDF

`SymbolScraper` is a tool for extracting characters and shapes from vectorized drawing instructions in PDF files, ignoring embedded images [38]. This requires identifying and extracting character shapes (*glyphs*) embedded in font profiles, as well as instructions for other graphics such as lines and polygons. Glyphs and drawing commands define how and where objects are drawn in a PDF. Drawing commands indicate a graphic type (e.g., for font characters, and straight vs. curved lines).

As seen in Fig. 4, graphic objects in PDF files are defined by instruction sequences. These terminate with an 'end-graphic' command (not shown). The instructions are in a postfix notation with arguments pushed on a stack before the operations that apply them. Note that coordinates in the JSON output shown in Fig. 4c do not match those in Fig. 4b, because the final line endpoints depend upon the line thickness and earlier context matrix. In a larger file, the context matrices are processed cumulatively.

PDF graphics are defined primarily by instructions for lines, rectangles, and Bezier curves. We use these as graphical primitives along with their parameters such as (x, y) points, line widths, whether objects are filled, etc. Graphical primitives are converted to *line strings* (polylines),[5] each of which is a sequence of straight line segments. We approximate Bezier curves in PDF as straight line segments, using

[5] Java Topology Suite: https://locationtech.github.io/jts/.

a parameter to limit the maximum distance that a point on the original curve can deviate from the approximated line segments, in points (i.e., 1/72 of an inch).

A small number of rules and additional parameters are used to extract the final input tokens (parameters shown in Table 1). Some straight lines are drawn as filled polygons, which are approximated by a line if the two longest lines cover more than a percentage of the polygon perimeter and have their angles within a small tolerance. Solid wedges (trapezoids) are identified in polygons based on the ratio of long:short side lengths. Positive charges are sometimes drawn with two overlapping lines tested for perpendicularity within an angular tolerance.

The final input tokens produced by SymbolScraper for the born-digital parser are bounding boxes, polygons, or polylines. Each have associated parameters, types, and labels.[6]

4.2 Minimum spanning tree (MST)

MSTs are widely used for constraints and optimization tasks involving point sets and other geometric object collections in continuous space (i.e., $\mathbb{R}^n$), including agglomerative clustering. For graphics recognition, MSTs have been used to constrain symbol and spatial relationship types when recognizing handwritten math formulas, e.g., by Matsakis [21] and Eto and Suzuki [8].

As can be seen in Fig. 1, chemical diagrams are even better suited to MST-based selection of spatial relationships than math formulas. The visual structure of math formulas may have as many as eight spatial relationship types, while molecule diagrams contain only one spatial relationship (`connected`). Symbols in formulas may be related at a distance, while connections in molecular diagrams are between neighboring symbols. Lines or other graphical objects that need to be combined into symbols (e.g., two parallel lines in a double bond) are also neighboring objects.

We construct an MST to connect graphical primitives with their nearest neighbor in a chemical diagram, breaking ties arbitrarily when two or more neighbors are equidistant. A complete undirected graph over all input PDF primitive pairs is generated first, with edges weighted by distance. By default, edge weights are the distance between the closest points on two objects; however, for line pairs we use their end-points to capture connection distances. This also prevents overlapping lines from having distance 0.

Invalid character connections are prevented by setting distances in our weighted adjacency matrix to ∞ when: (1) The absolute value of the cosine for the angle between characters falls between [0.1, 0.9], i.e., between [25.8, 84.3]°. This prevents (illegal) superscript or subscript character connections. (2) A line-character distance is more than 1.5 standard deviations from the mean line-character distance in the diagram. Pruning parameters are shown in Table 1.

We use Kruskal's algorithm to extract an MST with $n-1$ edges for n primitives, such that the sum of edge distances is minimal in the pruned adjacency matrix. An example MST over input graphics primitives is shown in Fig. 1b.

4.3 MST → visual structure graph

While an MST over PDF graphical primitives includes many connections needed to recognize molecular structure, connections often need to be added or removed. For example, an MST cannot contain cycles, and so we need to insert edges when three or more lines intersect. These and other changes are needed to produce the final graph capturing the visual syntax of a molecular diagram, e.g., as seen in Fig. 1d. The steps used for this transformation are presented below; parameters are shown in Table 1.

Negative Charges. We first distinguish negative charges from other lines. Lines are considered negative charges if they are: (1) roughly horizontal (0°), (2) no longer than a fraction of the average line length in the diagram, and (3) right adjacent to a character, with the line's vertical center in the upper half of the character's bounding box.

Restructure MST. Next we correct connections for 'floating' bond lines such as the double bonds in Fig. 1. These floating lines may not connect with their corresponding parallel line in the MST when another line's endpoint is closer. We consider creating an edge between a candidate floating line with degree 1 (one edge) in the MST with another nearby overlapping parallel line if it is within the five nearest neighbors of the line, and the average endpoint distances between the two lines is smaller than for the current neighbor. If so, the line is disconnected from its current neighbor and connected to the closer parallel line.

We then use distance-based clustering to add and remove connections based on MST distances.

1. *Line intersections.* Add missing non-parallel line intersections (e.g., for rings and multi-line intersections) where the lines' endpoints are within a ratio of the maximum distance between connected non-parallel lines.
2. *Character-line connections.* Filter MST char-line connection distances via Z-scores (i.e., standard deviations from the mean) before estimating the maximum char-line connection distance. Add all char-line edges within a ratio of this maximum distance.
3. *Split Floating Structures.* Prune edges with a distance larger than a ratio of a maximum distance. The connection type used to determine the maximum distance is selected in the following in order, based on first available distance

[6] Represented using the Python `Shapely` library.

Table 1 Parameters for PDF symbol parsing stages (see Fig. 3). For visual parsing of raster images (see Sect. 6) only tokenization is applied after creating a structured MST directly

Parameter (Value)	Primitive graph (MST)		3. MST → Visual Graph (a)	(b)	(c)
	1. Extract Symbols	2. Build MST	−ve Charges	Restr. MST	Tokenization
PDF GRAPHIC PRIMITIVES					
BEZIER_FLATNESS_PTS (0.25)	✓				
RECT2LINE_LONG_RATIO (0.85)	✓				
RECT2LINE_ANGLE_TOLERANCE (5.0)	✓				
ANGLES & PROXIMITY					
ANGLE_TOLERANCE_DEGREES (3.0)	✓		✓	✓	✓
CLOSE_NONPARALLEL_ALPHA (1.75)				✓	
CLOSE_CHAR_LINE_ALPHA (1.5)				✓	
SYMBOLS					
S-WEDGE_LENGTHS_DIFF_RATIO (0.7)	✓				
NEG-CHARGE_Y_POSITION (0.5)			✓		
NEG-CHARGE_LENGTH_TOLERANCE (0.5)			✓		
PRUNING EDGES					
ABS_COS_CHAR_PRUNE (0.1)		✓			
CHAR_LINE_Z_TOLERANCE (1.5)		✓			✓
MAX_ALPHA_DIST (2.0)					✓

type in the MST: (1) char-line distances, (2) parallel line distances, or (3) non-parallel line distances.

Tokenization. There are two steps for merging lines into bonds and characters into atom and group names: (1) merging adjacent characters and parallel lines, and (2) labeling bond types.

Merge Characters and Parallel Lines. Characters connected by edges are merged into text tokens, using the location of the nearest character as the connection point for a bond, if present (see Fig. 1d). Double bonds, triple bonds, and hashed wedge bonds are represented by adjacent parallel lines. Hashed and solid wedge bonds have a shorter side that begins the bond and a longer side that ends the bond, indicating the bond direction. Solid wedge bonds are trapezoids, while hashed wedge bonds are drawn as parallel lines of increasing length. All neighboring parallel line groups in the MST are merged, and annotated by the number of lines they contain. For example, in Fig. 1d, three pairs of parallel lines representing double bonds will each be merged and annotated with '2'.

Label Bonds in Line Groups/Wedges. Annotated line groups can then be labeled as *single*, *double*, or *hashed wedge* bonds by the number of lines they contain (i.e., 1, 2, or >3). Three parallel lines are a special case: both triple bonds and hashed wedge bonds may contain 3 parallel lines. We distinguish these by sorting the 3 lines topologically (i.e., top-down, left-to-right), and then determine whether these lines uniformly increase or decrease in size within the sorted list.

For wedge bonds, we need to identify new endpoints on the longest and shortest sides (for solid) or longest and shortest lines (for hashed) and restructure the final visual structure graph accordingly. Bond endpoints are important in the semantic analysis step, which we describe next.

4.4 Visual → molecular structure

In the final stage of the born-digital parser, visual structure is converted to molecular structure, and chemical information not directly visible in the diagram is added to produce a chemical graph. The chemical graph is then represented in a CDXML file capturing *both* visual and chemical structure. Note that this stage uses a deterministic process that involves no tunable parameters.

We first need to define explicit intersection points where line endpoints meet. These intersection points are defined by the midpoint between adjacent endpoints for connected lines in the visual structure graph. 'Hidden' carbon atoms are then inserted as nodes at bond line intersections, and at line endpoints without a neighbor. Nodes for bonds in the tokenized visual structure graph are removed, and replaced by edges labeled with the same bond type (see Fig. 1d, e).

CDXML Generation. CDXML is a file format representing molecules and reactions along with related text on a canvas or series of pages. For molecular data, both chemical structure

and the appearance of molecules on a 2D canvas are encoded in CDXML files. The format was created for the ChemDraw chemical diagram editor.

In CDXML tags define molecules, nodes (e.g., atoms, named groups), and bond connections in the diagram, along with annotations for node positions and appearance. We encode the locations of nodes on their associated page, so that the appearance and location of recognized molecules match the original document. Positions are also helpful with accurate conversion to other chemical formats (e.g., SMILES), and to capture spatial information in the chemical structure (e.g., for wedge bonds).

Annotate Names with Subgraphs: Molecules are often represented more compactly using chemical formulas or other names for substructures. For example, Fig. 1 shows an abbreviation node *NO*$_2$, a nitro group with an external connection available. We use a manually compiled dictionary of 612 common abbreviations with their associated subgraphs collected from the RDKit Python library,[7] ChemDraw, and our own work. For the abbreviation NO_2, we insert the full structure ($* \rightarrow N_1, N_1 \rightarrow O_1, N_1 \rightarrow O_2$) into the CDXML as a nested molecule 'fragment.' $*$ represents where the structure can be connected to other structures; O_1 and O_2 represents two oxygen atoms connected to the nitrogen N_1 through a single and double bond respectively.

5 Generating training data from visual graphs

In designing ChemScraper, we noticed that authors often copy molecular diagrams directly into their documents as raster images, which become embedded in PDFs. To create parsers for raster images with easily interpreted results, we require *explicit* correspondences between image regions and molecular symbols in generated visual structure graphs. Unfortunately, there is a shortage of training data with direct annotations of raster images. In addition to fast and accurate recognition, this was the second key motivator for creating our born-digital parser.

While one can create large datasets from SMILES using their rendered raster images, the correspondence between image regions and portions of SMILES strings is absent in such datasets. One can also generate molecular diagram images from MOL files, which include explicit molecular structure (e.g., atoms and their connection by bonds), along with optional 3d spatial positions. However, MOL files were not designed to describe image regions for characters, bonds, or other visual primitives in an image. For example, MOLs identify spatial locations of atom groups such as CH_3, but do not give the locations for its constituent `H` and `3` in an image.

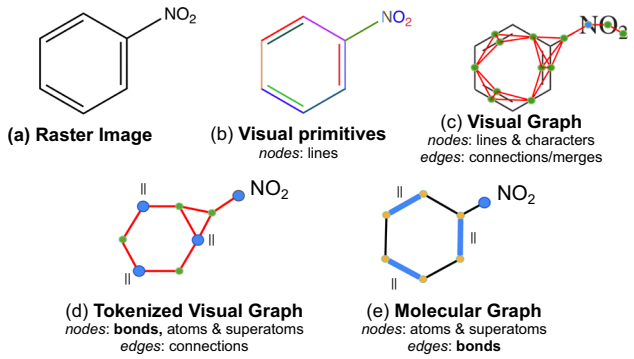


Fig. 5 Parsing Nitrobenzene ($C_6H_5NO_2$) from a raster image (**a**). **b** *Visual primitives.* The *N* is split into 3 lines. **c** *Visual Graph* extracted from visual parser. **d** *Tokenized Visual Graph* with merged nodes (bonds and named groups). **e** *Molecular Graph*. Blue nodes show the primitives of *N* merged into a character (**c**) and double bonds and atom/group names in (**d**, **e**). In **e** orange nodes are 'hidden' carbon atoms, and single/double bonds are converted from nodes to edges

A new data generation technique is required. First, we sought a stable visual primitive in pixel-based (raster) molecule images that would avoid merging symbols, and found that we could extract a type of line primitive reliably for this purpose (see Fig. 5b). Given the born-digital parse results for a molecule in PDF, we extract these line primitives from the rasterized PNG for the molecule, and align them with the PDF primitives based on maximum overlap.

The born-digital visual graphs annotated with line primitives can then be used for training models using the same line primitives as input. For these parsers, the visual primitive extraction replaces the first step of the born-digital parsing pipeline seen in Fig. 2, where rather than extract characters and lines directly, we may also extract image regions that over-segment (i.e., split) lines and characters.

Visual Primitives (Lines). From a raster image (PNG) for a PDF molecule rendered by the Indigo chemoinformatics toolkit,[8] we extract connected component (CC) contours, and convert these to polygons using a simplification algorithm (provided by `Shapely`). These polygons are transformed into a set of skeletal lines using pairs of adjacent parallel lines on the contour boundary. Each pair of parallel lines is replaced by their medial axis (i.e., line between the middle of the parallel lines' endpoints).[9] After the medial axis lines have been identified, pixels in CCs are segmented by assignment to the nearest axis line using a distance transform.

The resulting 'visual' line primitives can be seen in Fig. 5b. Some CC shapes such as curved lines and closed curves are unaltered by the process. The 2 is unsegmented because after

[7] https://www.rdkit.org.

[8] https://github.com/epam/Indigo.

[9] Parameters in Table 1 constrain angles and min. overlap.

identifying all skeletal lines for CCs in a molecule, to avoid segmenting small CCs, we test whether the average skeletal line length in a CC is less than the average for all skeletal lines. If this average length is smaller than the global average, we do not segment the CC. We also remove skeletal lines within CCs that are smaller than the global average skeletal line length, which avoids over-segmenting lines at dense intersections (e.g., at the connection point between two single bonds and a double bond). We split a long line in a triple or double bond by projecting the floating line onto it, and then testing if the overlap ratio r for the longer line is in the interval of one third to one half, with a margin of 10% ($\frac{1}{3} - \frac{1}{10} \leq r \leq \frac{1}{2} + \frac{1}{10}$).

For illustration, here we have manually broken the N into three parts; in practice, both characters and lines may be over-segmented. In Fig. 5b there are 15 visual primitives, versus 13 graphical primitives for the original PDF in Fig. 1a, b. 10 primitives are straight bond lines, and 5 primitives are for the characters in NO_2.

Visual Graph Generation. We now annotate raster images using our visual primitives and visual graphs before tokenization (see Fig. 1c) from our born-digital parser. We use Indigo to render PDFs from SMILES rather than PNG images as done in previous methods (e.g., MolScribe [30]). The born-digital parser is then run on the PDF images, and where the recognized SMILES and original SMILES match (i.e., the result is correct), we use the resulting visual graph as our preliminary ground truth data (e.g., see Fig. 1c).

We next assign visual line primitives to PDF graphical primitives in the born-digital visual graph. PDF images are converted to 256 DPI PNG images, and we extract visual line primitives as described above. The assignment of visual primitives to PDF primitives/symbols is determined by maximum overlap. In Fig. 5c, 1 line primitive is attached to each line node, 3 line primitives are attached to N, and one primitive is attached to each of the O and 2. Finally, we validate bonds between atoms against a MOL connection table generated from SMILES using Indigo.

To store visual graphs, we create label graph (Lg) files [24, 25] for both PDF primitives and visual line primitives. An example is shown in Fig. 6a. Primitives are represented by numeric identifiers and image contours, while typed objects are comprised of one or more primitives (e.g., `Single` bond: one line, character `N`: three lines).

A label graph file defines structure over declared primitives, using primitive groups (*objects*) and their relationships. In our label graph files, only `CONNECTED` relationships are explicitly defined, however `MERGE` relationships are defined implicitly between all primitive pairs in an object. In Fig. 6 `MERGE` edges exist between primitives 10, 11, and 12 for `N` (`Obj10`), and the connection between this character and the `Single` bond `Obj9` is represented by `CONNECTED` edges for (9,10), (9, 11) and (9,12). Similarly, all primitives in an object share a label (e.g., for `Obj10`, primitives 10, 11, and 12 are labeled `N`).

6 Visual parser

In Fig. 7 we present a multi-task neural network that parses raster images using the line primitives described in the previous section. The parser produces visual structure graphs, and is trained using our ground truth representation for raster images illustrated in Fig. 6. For formulas that contain `MERGE` edges, we use two versions of the input: (1) with no labels, relations, or `MERGE` edges defined (i.e., raw primitive input), and (2) with no labels or relations, but *all* ground-truth `MERGE` edges provided. This allows the model to learn more quickly how to classify symbols and relationships from whole objects rather than their parts.

This parser extends the LGAP model (Line-of-Sight Graph Attention Parser) [39] for parsing mathematical formulas. The parser creates visual structure graphs by generating labels for individual primitives and primitive pairs in an input graph, by classifying individual *queries*. Compared to the born-digital parser, the visual parser uses line primitives to replace the first stage of the pipeline in Fig. 3, and the visual parser replaces the second and third stage up to step 3(b) to produce a visual graph (restructured MST). The remaining tokenization and semantic analysis steps (steps 3(c) and 4) are unchanged.

Input. The parser input is a Line-of-Sight (LOS) graph over visual line primitives [7, 16], to prune edges between primitives that are 'blocked' by a primitive between them. In the LOS graph, edges are defined between primitives where an uninterrupted line may be drawn from the center of one primitive to a point on the convex hull of the other [20]. Connections and merges exist only between nearby primitives in molecular diagrams, as reflected by our use of MSTs in the born-digital parser. Here we prune LOS edges not within the $k = 6$ nearest neighbors of a primitive. There can be at most 4 lines or characters in a bond; we choose 6 neighbors to accommodate over-segmentation in visual primitives.

Features. Visual features are created by drawing line primitive contours directly into 28×28 binary images for (1) individual primitives (node queries), (2) primitive pairs (edge queries), and (3) context images containing the $k = 6$ nearest neighbors centered around each query (one per node/edge query). Query and context images are passed separately through a single SE-ResNext backbone producing 32 feature maps per image [15, 46]. The first layer of the SE-ResNext encoder is modified, replacing the 7×7 convolutional kernel by 3×3, using a stride of 1, and `same` padding. We also

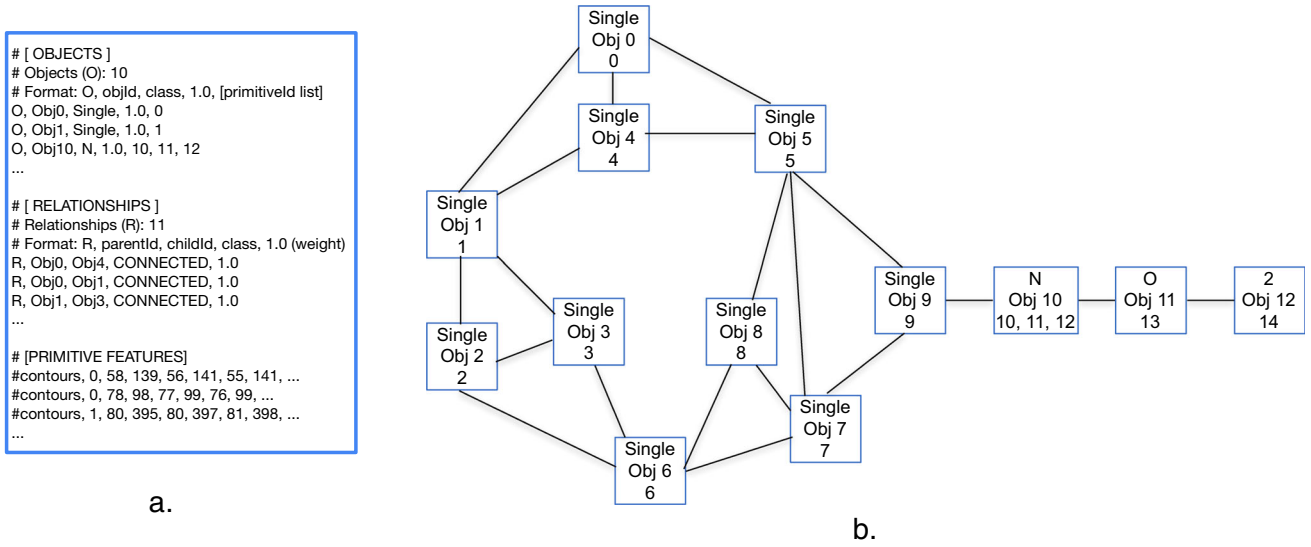


Fig. 6 Ground truth visual graph generated for Fig. 1c. **a** Label graph file with Objects (`O`), Relationships (`R`) and Visual primitives with contour points (`#contours`). **b** Visualization showing primitive identifiers, node labels, and edges (all edges labeled as `CONNECTED`). Objects for single bond contain one line primitive each, while the character `N` contains three line primitives. A second file is created using 13 PDF primitives (vs. 15 visual line primitives shown here)

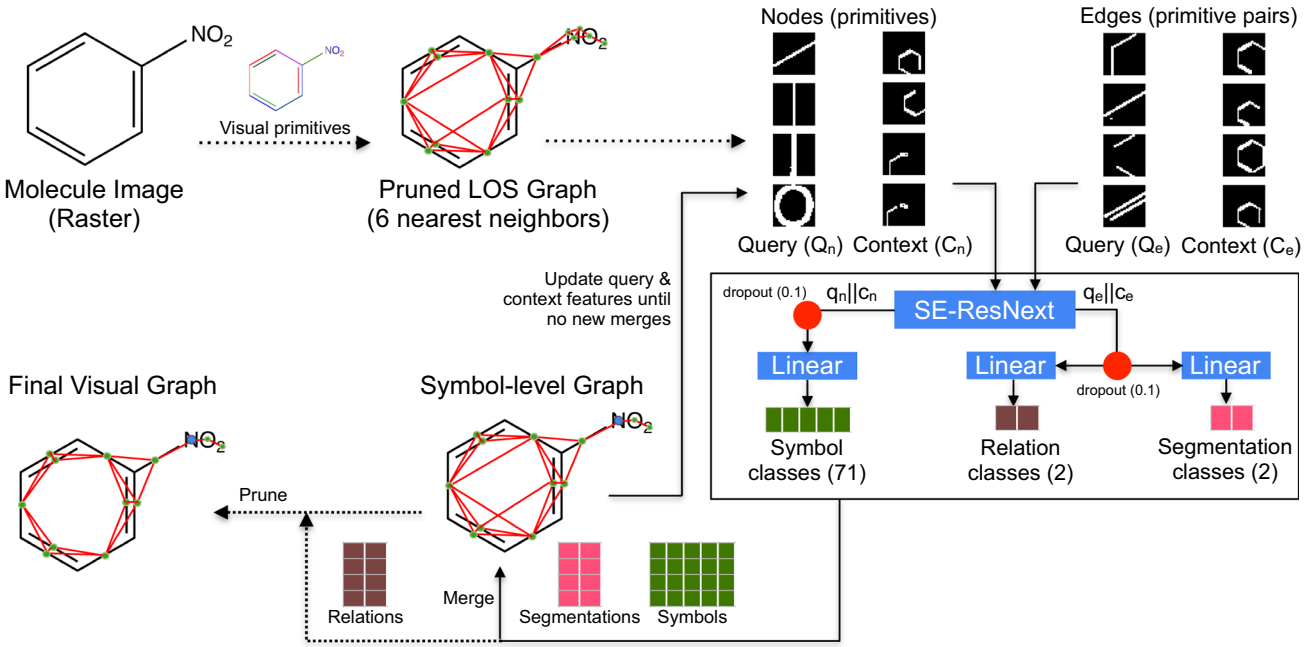


Fig. 7 Parsing a raster image of nitrobenzene ($C_6H_5NO_2$). Line contours are extracted as primitives, over which a pruned LOS graph is built. At top-right, four node and four edge queries are shown, at bottom-left their classification tensors (rows: queries, columns: classes). (Q)uery and (C)ontext features enter an SE-ResNext block. Two-layer Multi-Layer Perceptrons (MLPs) estimate probabilities for symbol, segmentation (MERGE), and relationship (CONNECTED) probabilities. Merges are applied (e.g., for 'N'), with symbol/relationship probabilities averaged across primitives. The model runs recurrently, updating queries and their contexts until no new merges are found (e.g., two passes for this example)

remove the first maxpool layer because feature images are small.

Feature maps are average pooled in 7 pyramidal regions (image, 3 vertical, 3 horizontal). The final query visual features are the pooled convolution responses for a node/edge and its associated context (i.e., $q_n||c_n$ or $q_e||c_e$). For 32 features maps with 7 average-pooled regions, the query and context images produce $2 \times 224 = 448$ features. We add three positional encodings to query vectors in the form of bounding boxes (BBs) $(x_{min}, y_{min}, x_{max}, y_{max})$ with coordinates normalized to be percentages of width/height:

1. Query BB relative to the formula window
2. Query BB relative to the context window
3. Context window BB relative to the formula

For edge queries, we use the combined primitive pair position as the query position. Adding these three BBs each query vector contains $448 + (3 \times 4) = 460$ features. Dropout is applied for regularization (rate of 10%).

Classification. As seen in Fig. 7, node and edge queries are classified using three two-layer multi-layer perceptrons (MLPs):

1. Node symbol class (71 class)
2. Edge primitive merge (2 class)
3. Edge primitive connection (2 class)

For each classification, a hidden linear layer (512 units) is fully connected to the class output layer. For node queries the 71 classes include digits, characters, charges (+,-), parentheses, and straight lines. Edge queries are classified twice, once to identify whether a primitive pair belongs to the same symbol (MERGE), and then to test whether the primitive are from two connected objects in the diagram (CONNECTED).

Recurrent Execution. The parser segments symbols bottom up from input primitives, updating query and context images during recurrent execution. Execution is performed recurrently until edge queries classified as MERGE with probability > 0.5 are unchanged from the previous pass (i.e., a fixed point is reached). On a recurrent execution, query images, context images, and positional encodings are all updated for merged primitives. Merges are identified by connected components along MERGE edges.

Note that this is not a conventional recurrent neural network (RNN) where a state vector is updated across executions. Instead, we simply update input features directly as the segmentation changes. For example, an N broken into three primitives may be merged in the second pass to produce three node queries containing all three primitives. This allows the N to be classified in a single query, rather than in three parts within the first iteration. Here the query images are identical for each merged primitive, but note the context image for the primitives will differ because they are centered on the original input primitive associated with each query. This addresses class imbalance by representing multi-primitive symbols multiple times, each with a slightly different context image.

Recurrent execution stops when no change in MERGE decisions is identified. Edges identified with a probability of being CONNECTED > 0.5 are selected; any edges not selected for MERGE or CONNECTED are removed. Symbol and relationship probabilities are then computed by averaging them across primitives in segmented symbols and their connections.

Training. Random over-sampling of node queries is used to balance edge and node queries. To balance positive and negative edge examples, we randomly over-sample positive edge examples (MERGE and CONNECTED), so that each have the same number of positive and negative examples.

Node and edge queries are processed together using a batch size of 64. The sum of cross-entropy losses (X) for node and edge queries computed for each batch is

$$\sum_{q_n \in Q_n} X_S(q_n) + \sum_{q_e \in Q_e} X_M(q_e) + X_C(q_e) \quad (1)$$

where X_s, X_m and X_r are the cross entropy loss given the correct target response vectors (1-hot) and softmax distributions for (S)ymbol classification, primitive (M)ERGE, and primitive (C)ONNECTED outputs. For backpropagation, we use an Adam optimizer with learning rate 0.0005, β values of (0.9, 0.999), and no weight decay.

7 Evaluation

We next evaluate the accuracy of our parsers. It is important to remember that the ChemScraper born-digital parser utilizes PDF information for characters, lines, and other graphical objects that parsers working from raster (pixel) images do not. Our analysis includes a graph-based analysis of recognition errors at the level of molecule structure present that provides information missing in standard SMILES-based evaluation methods.

Datasets. For tuning born-digital parser parameters and generating visual parser training data, we use 5000 molecules (46 unique SMILES characters) extracted from PubChem[10] prepared by the MolScribe team [30]. For benchmarking, we use three datasets: (1) the USPTO synthetic dataset with 5179 PNG images generated by the Indigo toolkit from SMILES strings (37 unique SMILES characters) [31], (2) UoB (5740

[10] https://pubchem.ncbi.nlm.nih.gov.

molecule PNG images + SMILES: 33 unique characters [35]), and (3) CLEF (992 molecule PNG images + SMILES: 71 unique characters [29]).

The born-digital parser is run on Indigo-rendered PDFs from SMILES ground truth, including for the UoB and CLEF datasets. For the USPTO synthetic set, the rendered PNG and PDF images are essentially identical, but this is not true for the CLEF and UoB data sets where scanned images of molecules were annotated with SMILES; in this case rendering the SMILES using Indigo may produce images in different styles, fonts, and orientations than the scanned molecule images.

Additionally, as described in Sect. 5, we generate annotated visual graph data for training our visual parser that recognizes from raster images. This comprises 3416 label graph files from the original pool of 5000 molecules sourced from PubChem that could be accurately converted into exact SMILES strings. Errors include 240 diagrams misrecognized from valid visual primitives by the born-digital parser, and 1344 diagrams with errors produced in primitive extraction, alignment, and converting visual graphs to SMILES strings. This training dataset includes molecules represented by 32 unique symbol classes. A limitation is that there are test set symbols missing in this training set. For the USPTO dataset 4 symbols are absent (`1, a, D, b`), from CLEF 26 symbols are absent (including `*, R, X, 0`), and from the UoB dataset 2 symbols are missing (`:, 0`).

Implementation/Systems. SymbolScraper is built in Java using Apache's `PDFBox` and the `Java Topology Suite`, while the ChemScraper born-digital parser is implemented in Python using the `Shapely` (2d geometry), `networkx` (graphs), `numpy`, and `mr4mp` (map-reduce) libraries. The ChemScraper born-digial and visual parsing pipelines are Python-based, along with the visual line primitive extractor.

Born digital parsing runs were made on a Ubuntu 20.04 server, with a Intel(R) Xeon(R) CPU E5-2667 v4 (3.20 GHz) and 512 GB RAM. Experiments for the visual parser were run on another Ubuntu 20.04 server with hard drives (HDD), an A40 (48GB) GPU, a 64-core Xeon Gold 6326 (2.9 GHz), and 256 GB RAM.

7.1 Representations and metrics

We describe the molecule representations and associated metrics used in our evaluation below.

SMILES strings: matches and similarity Simplified Molecular-Input Line-Entry System or SMILES [45] represents molecules by the sequence of atoms seen in a traversal of the molecular structure graph. SMILES are compact, and readable for domain experts. ChemScraper-generated CDXMLs are first translated to SMILES using ChemAxon's `molconvert` tool. After this, we canonicalize both CDXML and benchmark SMILES to remove differences in their atom order, which can vary for the same molecule. SMILES canonicalization is performed using the RDKit library via the function `CanonSmiles()`, with `ignore_chiral=False`.

SMILES strings are compared by (1) the percentage of exact matches, and (2) the *inverse* of the average Normalized Levenshtein Distance (NLD). The *levenshtein distance* is the minimum number of insertions, deletions, or substitutions needed to convert one SMILES string to the other [37]. The distance is normalized to [0, 1] using the minimum/maximum possible edits based on the SMILES string lengths. The inverse of the average NLD is given by subtracting the average NLD from 1, giving a *similarity* in [0, 1], with 1 produced for identical SMILES strings.

Limitations. Molecular formulas are naturally represented as graphs, where atoms and bonds have well-defined relationships and spatial arrangements. In contrast, SMILES representations are linear character strings *describing* graph structure. These SMILES characters have no direct connection with the atoms and bonds present in an input image (i.e., where atoms appear is not represented).

Levenshtein distances for SMILES strings may correspond to multiple operation sequences of the same length. In this case, Levenshtein-based SMILES metrics do not uniquely identify which parts of the input are incorrectly recognized. It is thus tempting to instead use graph edit distances over molecule structure graphs directly, with operations that insert/delete/relabel nodes and edges. Unfortunately, this can also result in ambiguous minimal edit sequences, and errors may again not be uniquely identified.

The main issue here is a missing correspondence between input image regions and the nodes/edges in a molecular structure graph representation. If molecular structure graphs include input image locations (e.g., bounding boxes) their nodes may be aligned spatially and then compared using adjacency matrices. We describe the first application of this approach to chemical structure recognition evaluation next.

Labeled graphs for molecular structure: label hamming distance and similarity Example molecular structure graphs are shown in Figs. 1e and 5e, which are equivalent.[11] For the ChemScraper parsers, molecular structure graphs produced using born-digital primitives (see Fig. 1b) or visual primitives (see Fig. 5b) contain polygons representing the image locations for hidden carbons and atom/group labels. We use these graphs directly for evaluation.

Labeled graphs defined over the *same* nodes with known input locations can be directly compared using their adja-

[11] *Note* The graphs are mostly undirected, but wedge bonds going 'in'/'out' of a page require directed graphs.

cency matrix entries. Recognition errors are easily identified by differing labels in adjacency matrix cells, and located within an input image using the node locations. With a particular bottom-up representation for grouping nodes (i.e., segmentation), errors may be identified even when node groupings disagree, or nodes are missing in one or the other graph [50].

Handwritten math formula recognition was evaluated in this manner for the early CROHME competitions, with ground truth and recognizer outputs defined over the same handwritten strokes [24]. The LgEval library[12] was used to compute metrics and visualize errors [24, 25, 38]. One can view all errors using the `confHist` tool including missing nodes and relationships. Repeated errors for nodes, edges, and subgraphs are compiled in histograms that may be explored in HTML pages.

Here we take a slightly different approach. Rather than graphs sharing nodes, corresponding ground truth and output nodes in molecular structure graphs are aligned (i.e., assigned the same identifier) based on spatial overlap in a PDF image. After this alignment, we apply the same adjacency matrix-based evaluation metrics and tools used for CROHME.

We first assign identifiers to nodes in the ground truth graph, which are atoms or named groups (e.g., SO_2) and hidden carbons at line intersections. We have adapted MolScribe code to locate atom/group names and hidden carbons in a PDF image for a molecular diagram generated using Indigo. Then, parser output graph nodes are given the identifier of the ground truth node that they have maximum overlap with, breaking ties arbitrarily. Where multiple output nodes overlap one ground truth node, or an output node does not overlap a ground truth node (e.g., missed line intersections produce extra hidden carbons), additional unique identifiers are created. Bonds are then defined using labeled edges between nodes using these bond types: (`single, double, triple, wavy, solid wedge, hashed wedge`).

After alignment, adjacency matrices are used to identify *all* structural differences from the labels in corresponding cells. Both rows and columns of adjacency matrices for: (1) ground truth, and (2) parser output, are labeled by the node identifiers obtained during alignment. Node labels are located in diagonal entries (e.g., (n_1, n_1)) and edge labels are provided in the off-diagonal entries (e.g., (n_1, n_2)). For nodes, we compute the percentage of ground truth nodes aligned with an output graph node with the same label (i.e., (R)ecall), and the percentage of output nodes aligned with an identically labeled ground truth node (i.e., (P)recision). We combine Recall and Precision using their harmonic mean F_1:

$$F_1 = \frac{2RP}{R+P}.$$

We also report the analogous F_1 measure for edges (bonds). An output edge is correct if its end nodes and label match ground truth. Finally, we report the percentages of molecules with correct structure (i.e., correct `MERGE` and `CONNECTED` relationships), and with both correct structure and node labels.

Table 2 Grid search parameters

1. ANGLES & PROXIMITY	
`ANGLE_TOLERANCE_DEGREES`	{1, 3, **5**, 10, 15}
`CLOSE_NONPARALLEL_ALPHA`	{1, 1.25, 1.5, **1.75**, 2.0}
`CLOSE_CHAR_LINE_ALPHA`	{1, 1.25, **1.5**, 1.75, 2.0}
2. SYMBOLS	
`S-WEDGE_LENGTHS_DIFF_RATIO`	{0.70, 0.85, **0.90**, 0.95}
`NEG-CHARGE_Y_POSITION`	{0, **0.25**, 0.5}
`NEG-CHARGE_LENGTH_TOLERANCE`	{0.33, **0.5**, 0.66}
3. PRUNING EDGES	
`ABS_COS_CHAR_PRUNE`	{0.10, **0.15**, 0.20}
`CHAR_LINE_Z_TOLERANCE`	{1.0, **1.5**, 2.0}
`MAX_ALPHA_DIST`	{2.0, **2.5**, 3.0}

Values tested are shown, with default values in bold

7.2 SMILES-based evaluation

Parameter Tuning and Rendering. Each molecule in our 5,000 PubChem molecules for parameter fitting was rendered with Indigo using 3 randomly selected parameters. The rendering parameters are described below. For benchmarking the born-digital parser, we use the Indigo default rendering parameters. This is done to insure PDF molecules for the born-digital parser have the same appearance as PNG images in the USPTO dataset, which is our primary collection for benchmarking.

The final parameter values seen earlier in Table 1 are obtained using grid search, with the exception of the PDF GRAPHICS PRIMITIVES group belonging to `SymbolScraper`. To keep the tuning process manageable, we divided the grid search into 3 stages, one per group in the order given in Table 1. Initial default values were identified. After each parameter group's grid search was complete, learned values replaced the default values. Value ranges and defaults are shown in Table 2.

We also tested the effect of the MST pruning parameters discussed in Sect. 4.2: removing them harms accuracy. For the USPTO dataset removing the absolute cosine angle threshold for characters produces 93.72% SMILES matches, removing the threshold for line-character distances produces 97.06% SMILES, matches and removing both produces

[12] https://gitlab.com/dprl/lgeval.

Table 3 Molecular structure recognition benchmarks

Models		SYNTHETIC IMAGE	*SCANNED IMAGE	
		USPTO (5719)	CLEF-2012 (992)	UoB (5740)
Rule-based	MolVec 0.9.7	95.40	83.80	80.60
	OSRA 2.1	95.00	84.60	78.50
	Imago 2.0	–	68.20	63.90
Neural Network	Img2Mol	58.90	48.84	78.18
	DECIMER	69.60	62.70	88.20
Graph Outputs	OCMR	–	65.10	85.50
	SwinOCSR	74.00	30.00	44.90
	Image2Graph	–	51.70	82.90
	MolScribe	**97.50**	88.90	87.90
	MolGrapher	–	**90.50**	**94.90**
ChemScraper	**Born-Digital Parser (PDF input)**			
	(PDF rendering errors)	(15) **98.16**	*(71) 89.32*	*(0) 94.41*
	*Skipping rendering errors	**98.42**	*96.20*	*94.41*
	Visual Parser (PNG input)	85.02	–	–

Percentages of generated SMILES matching ground truth are shown. For USPTO both PNG and PDF images are rendered using Indigo, but rendered SMILES PDFs may differ from scanned PNGs for CLEF and UoB (indicated by italics)

93.20% matches. Including the pruning parameters produces 98.16% exact SMILES matches.

Benchmarking: Born-Digital Parser. Table 3 compares ChemScraper and existing molecule parsing models. For the USPTO dataset, we see that the born-digital parser obtains the highest rates. Note that the 'rendering failure' for USPTO applies to all systems, because the SMILES for these 15 molecules are missing in the collection itself. Given this, the born-digital parser working from PDFs outperforms the neural models working from raster images by nearly 1%, and rule-based system working from raster images by roughly 3%. The strong performance of the born-digital parser is because of the additional information available from PDF instructions, and the robust design of the born-digital parser.

The model also obtains competitive rates for CLEF and UoB, but note that this is for Indigo-rendered SMILES, and not the provided PNGs because PDF images are not provided in these collections.

In terms of execution time, running the born-digital parser on the USPTO-Indigo dataset (5,719 molecules) with a single process took 28.01 mins (293.39 ms/formula), i.e., 3.4 molecules/sec, with a peak CPU memory use of 230 MB. With multiple processes (32) the total time is reduced to 1.81 mins (19.04 ms/formula), i.e., 52.5 molecules/sec. Performance benchmarks from Rajan et al. [31] show that on a Linux workstation with Ubuntu 20.04 LTS, two Intel Xeon Silver 4114 CPUs and 64 GB of RAM, processing the USPTO-Indigo dataset took 28.65 min for MolVec 0.9.7, and 145.04 min for OSRA 2.1. Thus, on comparable systems, our born-digital parser operates at similar or faster speeds compared to other rule-based methods.

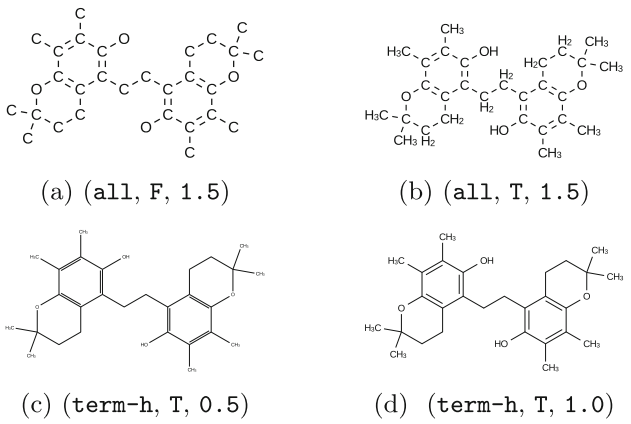


Fig. 8 Rendering a molecule with different parameters (Indigo toolkit). Each of **a–d** indicate the label mode, whether implicit hydrogens are shown, and the relative thickness. Parameters in **d** are the defaults. The born-digital parser recognizes all four versions correctly

Rendering: Sensitivity Analysis. To check the robustness of the born-digital parser, we used the rendering parameters of Indigo to perform a sensitivity analysis. We tested three rendering parameters visualized in Fig. 8. Parameters/values considered are:

1. `relative-thickness`: Boldness of graphic and text objects. Values considered: {0.5, 1, 1.5}. The default is 1.

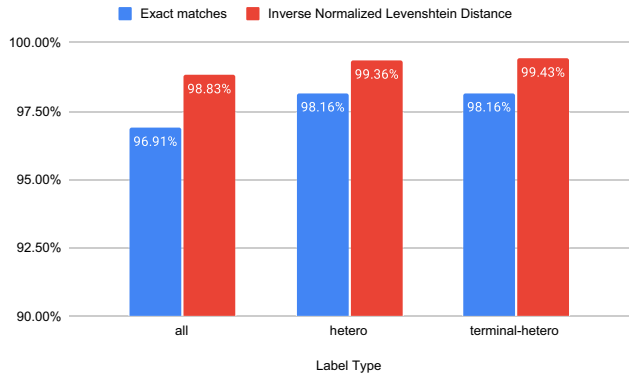


Fig. 9 Sensitivity of Born-Digital Parser to Label Rendering Parameter. SMILES-based evaluation is used. Other parameters have default values, with `render-implicit-hydrogens-visible` as True and `render-relative-thickness` to 1

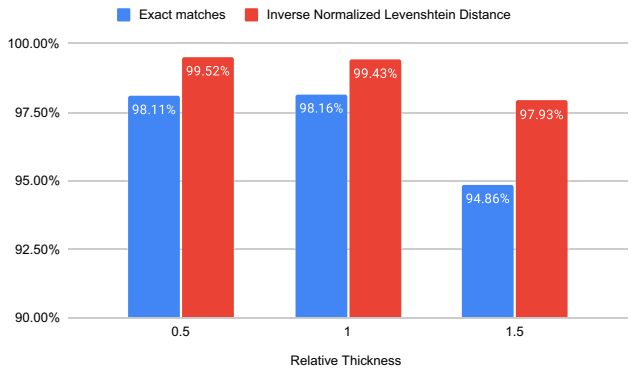


Fig. 10 Sensitivity of Born-Digital Parser to Thickness Rendering Parameter. Higher thickness reduces accuracy. Other parameters: `render-implicit-hydrogens-visible` is True, `render-label-mode` is terminal-hetero

2. `render-implicit-hydrogens-visible`: Whether to show implicit hydrogens. Default is True.
3. `render-label-mode`: Which atom labels to show: {hetero, terminal-hetero, all}. *all* shows all atoms. There is a *none* option we omit because it leads to ambiguous molecules. Default is *terminal-hetero*.

This produces 18 parameter combinations for rendering. We evaluated our parser with each of them for the USPTO Indigo dataset, using SMILES matches and inverse normalized levinshtein distances for evaluation.

Figure 9 shows how different atom labelings affect performance of the parser. Including all atom labels slightly hurts performance, in part because the more dense a molecule becomes, the more probable it is for the parser to connect atoms incorrectly. Figure 10 then shows the effect of rendering with different thicknesses. Lower thicknesses produce stronger results, again because this decreases the density of the molecule. As seen in Fig. 8, lower thickness increases the distance between unconnected objects.

Figure 11 compares performance when rendering molecules with or without implicit hydrogens. The difference between the conditions is minimal, with 14 fewer exact matches (roughly 0.06%) than when showing implicit hydrogens. This difference is due to merging errors of different groups that are close, similar to the crowding of Fig. 8b.

Overall, the born-digital parser is quite robust to these changes in rendering parameters. This robustness was achieved by gradually increasing the reliance of the born-digital parser on graph properties while reducing the number of parameters used; additional reductions in parameters are likely possible.

Benchmarking: Visual Parser. For the synthetic USPTO dataset, our visual parser trained using outputs from our born-digital parser, obtains a recognition rate of 85.02%. While this rate is lower than that seen for transformer-based

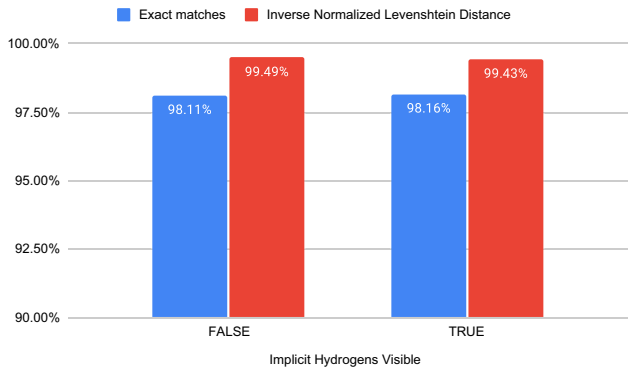


Fig. 11 Sensitivty of Born-Digital Parser to Showing Implicit Hydrogens. Other parameters: `render-label-mode` is terminal-hetero and `render-relative-thickness` is 1

methods like MolScribe [30] and rule-based methods such as MolVec and OSRA [9], this result still demonstrates potential. Notably, MolScribe is trained on 1.68 million examples with various chemical structure-based and image-based augmentations, and employs a SWIN transformer model with 88 million parameters. In contrast, our visual parser was trained on a much smaller dataset of 3,416 annotated images, without augmentation, and using a simpler SE-ResNeXt model with 4 million parameters. Despite these differences, our parser outperforms SWIN-OCSR [47], which also uses a SWIN transformer but is trained on 4.5 million molecules.

We have omitted results for the real datasets (CLEF and UoB) due to limitations in our initial training dataset, which is missing symbols from these sets and training using a single set of Indigo rendering parameters as mentioned earlier. This first training set does not adequately capture the diverse styles and structural variations seen in the non-synthetic data sets. We will address this in future work. We will note here however, that the visual line primitives extracted from the real images are accurate.

We conducted training runs on the Pubchem dataset, which consisted of queries for 3,416 molecules in three forms: primitives, whole symbols, and symbols detected during training. Each epoch averaged 155.6 min, with the model completing 19 epochs in about 49 h. This training time is notably shorter than other systems, such as DECIMER [32], which required 27 days to converge on 15 million structures, demonstrating efficiency with fewer data to achieve comparable results.

However, testing on the synthetic USPTO dataset (5,719 molecules) took 18.6 h (11.74 secs/molecule), which is slower compared to systems like MolGrapher [23] and OCMR [44] that process a single molecule in less than a second. The slow inference time is due to inefficiencies in our first implementation. In particular, re-assembling query outputs for formulas and writing visual graphs are currently slower than they could be. Future versions will accelerate these components.

7.3 Graph-based evaluation

For fine-grained evaluation of ChemScraper, we require molecule graph representations for both ground truth and the predicted molecules. Given we have already created chemical structure graphs subsequently converted to CDXML format, we can readily employ these graphs for evaluation. It is important to note that the molecular graphs utilized for evaluation differ from the visual graphs created in Sect. 5 to annotate raster images.

Molecular Graphs for Evaluation. The predicted graph corresponds to the final stage in the parsing algorithm, shown in Fig. 1e. These graphs are generated in the final step of the born-digital parsing pipeline (see Fig. 3). This graph assumes the representation of atoms or atom groups as nodes, with edges representing bond types associated with nodes, which may have one of the following types: `{Single, Double, Triple, Solid Wedge, Hashed Wedge}`. To construct a ground truth molecular structure graph, we use a `MOL` object generated by Indigo from the corresponding SMILES representation. We then extract atom positions along with the adjacency matrix for bonds between atoms using MolScribe code [30] with minor modifications.

We identify correspondences between nodes in parser output and ground truth graphs using atom coordinates from Indigo (ground truth) and Symbol Scraper (parser output). Minor discrepancies in atom coordinates are resolved using minimum distances between corresponding atom pairs. Corresponding nodes are giving the same identifiers.

Finally, we create object-relationship label graph files (Lg files) as described in Sect. 5. 'Object' entries represent individual atoms or atom groups, and the 'Relationship' entries denote bond edges with bond type labels between the atoms, as opposed to specifying the type of connections between visual elements.

Analysis: Born-Digital Parser. We use `LgEval` to compare molecular graphs to obtain the metrics in Table 4. The table shows a disparity between recognition rates when using labeled graphs (last column) vs. the exact SMILES matches shown in Table 3. This arises because SMILES string-based metrics lack sensitivity to direction and errors for 3D bonds, such as hashed and solid wedge bonds. In this way, SMILES exact matches may be misleading in terms of identifying correct molecular structures. In contrast, our graph-based metrics readily identify such errors.

Table 4 shows a large decline in recognition rates when using the hardest rendering condition for the parser, despite only a 0.83% reduction in accurate detection of edges in molecular graphs. This is mainly due to the intricate network of edges and relationships, particularly in large structures with rings. Even a 1% error in relationships, as seen in the USPTO-Indigo dataset with 382,058 target relationships for 5,719 molecules, substantially affects accuracy.

In the `confHist` tool error summary (an excerpt is shown in Fig. 12), common errors for the default rendering include missed single and triple bonds. The run for the hardest rendering parameters produces a notable increase in the count for the most frequent errors, including missing single and hashed wedge bonds. This unexpected difficulty with easier-to-detect bonds is due to the density of molecules in the hardest rendering condition, which produces short bond lines and a compact structure (See Fig. 8b). This poses challenges for our graph transformations using thresholds to accurately detect bonds or establish correct connections between entities. This illustrates where greater use of visual features may be beneficial within the born-digital parser itself.

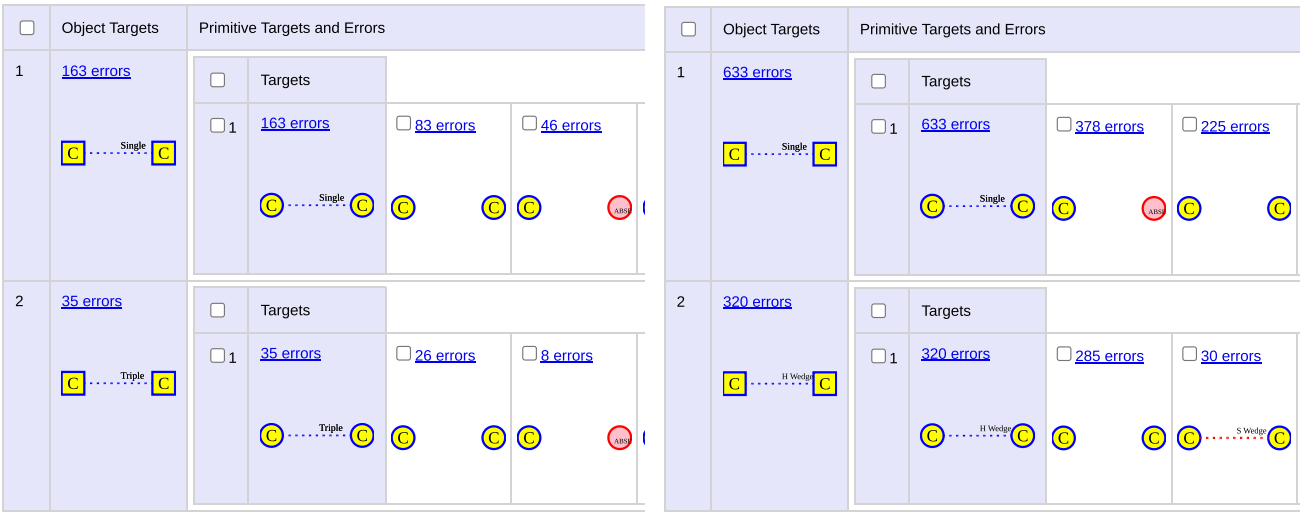
Analysis: Visual Parser. For molecular diagrams produced by the visual parser for USPTO, symbols including different characters, numbers, and wedges are often misclassified as `Single` bonds. This is mainly due to class imbalance in the training data that predominantly features `Single` lines (roughly 70% of symbols in training are single lines). Errors also include incorrect segmentations, particularly for characters like `N`, and `H` that are frequently over-segmented. This is also likely due to their rarity in the training data. Additionally, relationship errors, notably missed connections between lines and characters, are comparatively more common due to the predominance of line-line connections over line-character connections.

The class imbalance in symbols and relationships, especially the predominance of the `Single` class and line-line connections, highlights the need for better recognition of less frequent classes to improve the parser's performance on diverse molecular structures. Additionally, the training set does not include all symbols present in the test sets,

Table 4 Born-digital parser label graph metrics for different rendering parameters (5719 molecules)

Render	Rendering Parameters			Correct node (labels) F_1	Correct edge (labels) F_1	Molecules Struct.	+Class
	label_ mode	implicit_ hydrogens_ visible	relative_ thickness				
Default	Terminal-hetero	True	1	99.96	99.84	98.49	97.62
Hardest	All	True	1.5	99.65	99.01	81.89	81.12

Shown are F_1 measures for symbol labels, correct labels, and complete graphs



(a) Default Rendering Parameters

(b) Hardest Rendering Parameters

Fig. 12 Relationship Confusion Histograms for Renderings in Table 4 (truncated at right for space). Hyperlinks show molecules with specific errors, check boxes allow selecting molecules with errors for export. *Default rendering:* the top 2 errors are missing single and triple bonds. We can observe that in both cases, at times a missing (ABSENT) hidden carbon is the cause. *Hardest rendering:* missing single bonds are again the most frequent error, caused half of the time by a missing carbon. The second most-frequent error is missing hashed wedges between carbons, where no bond is detected, or because of misclassification of hashed wedges as solid wedges

which impacts the parser's ability to accurately recognize and interpret a full range of molecular symbols. Addressing this imbalance and coverage is important for future enhancements.

8 Conclusion

We have introduced the ChemScraper born-digital molecular diagram parser, along with improved extraction for characters and graphics from PDF (SymbolScraper). To address a shortage of training data for molecular diagrams in raster images, we use the born-digital parser to annotate raster images with visual structure graphs. This data is used to train a visual parser for raster images that uses a novel multi-task neural network run recurrently. Both the born-digital and visual parsers produce molecular structure graphs in CDXML which can be used with well-known chemical drawing tools (ChemDraw, Marvin) and easily converted to other molecular structure representations (e.g., SMILES, MOL, and InChI).

We also apply the adjacency matrix-based evaluation metrics developed for CROHME to molecular diagrams. These metrics and the LgEval tools offer a detailed assessment of parser performance, and identify bond structure errors missing in conventional SMILES-based evaluation.

Limitations of this work include:

1. Images considered are noise-free vector and rasterized-vector images from a single rendering model (Indigo) created using a limited set of parameters. While modern PDFs contain relatively clean images, noisy images (e.g., scans of older documents) would require modified image primitives, annotation strategies, and parser designs.
2. Born-digital parser parameters may be improved with larger grid searches, Bayesian optimization, and using visual features.

3. Graph transformations are manually defined; learned transformations may be more robust.
4. Our first visual parser has slow inference and does not yet generalize well to real images, due to limited class coverage and variation in our first training dataset.

Opportunities for future work include:

1. PDF primitives extracted by `SymbolScraper` provide high-precision locations for text and graphics. This can be applied in extraction, search, and visualization applications.
2. Developing a more domain-agnostic technique for born-digital parsing. Perhaps GNNs, graph rewriting systems, or encoder–decoder models could improve results obtained from `SymbolScraper` output.
3. The visual parser and graph-based evaluation methods are not domain-specific, and could be applied to other graphics including mathematical formulas and tables.
4. Applying the presented techniques to index molecules and other graphics in PDF collections for graphics-aware search applications such as MathDeck [1]. This was the original motivation for this work, and something that we are eager to pursue.

Acknowledgements This work was supported by the National Science Foundation USA (Grant #2019897, Molecule Maker Lab Institute). We thank Matt Langsenkamp, Matt Berry, Kate Arneson, and other members the NCSA team who helped create the online ChemScraper system.

Author Contributions S, A. K. refactored the system, enhanced some functionalities and wrote around 35% of the paper. D, A wrote around 20% of the paper and coded the first version of the system. A, B wrote around 15% of the paper, added some functionalities, coded most of the evaluation. C, M wrote around 20% of the paper and refactored the first version of parser. O, B wrote around 10% of the paper, provided chemist related information and feedback. D, S provided data. Z, R helped refactor the system, lead the research, rewrote and organized the paper.

Data availability The training data generated and used in this study is publicly available and can be accessed at https://www.cs.rit.edu/~dprl/data/icdar2024/. The dataset generation script is provided in the repository code provided in the Introduction section above.

Declarations

Conflict of interest The authors declare no competing interests.

References

1. Amador, B., Langsenkamp, M., Dey, A., Shah, A.K., Zanibbi, R.: Searching the ACL anthology with math formulas and text. In: ACM SIGIR Conference on Research and Development in Information Retrieval, pp. 3110–3114 (2023). https://doi.org/10.1145/3539618.3591803
2. Bristol-Myers Squibb—molecular translation competition, Kaggle (2021). https://www.kaggle.com/c/bms-molecular-translation
3. Baker, J.B., Sexton, A.P., Sorge, V.: A linear grammar approach to mathematical formula recognition from PDF. In: Carette, J., Dixon, L., Coen, C.S., Watt, S.M. (eds.) 16th Symposium on Intelligent Computer Mathematics, LNCS, vol. 5625, pp. 201–216 (2009). https://doi.org/10.1007/978-3-642-02614-0_19
4. Bukhari, S.S., Iftikhar, Z., Dengel, A.: Chemical structure recognition (CSR) system: automatic analysis of 2D chemical structures in document images. In: International Conference on Document Analysis and Recognition (ICDAR), pp. 1262–1267 (2019). https://doi.org/10.1109/ICDAR.2019.00-41
5. Campos, D., Ji, H.: IMG2SMI: translating molecular structure images to simplified molecular-input line-entry system (2021). arXiv:2109.04202
6. Comelli, P., Ferragina, P., Granieri, M.N., Stabile, F.: Opt. Recognit. **44**(4), 627–631 (1995)
7. de Berg, M., Cheong, O., van Kreveld, M., Overmars, M.: Computational geometry. In: de Berg, M., Cheong, O., van Kreveld, M., Overmars, M. (eds.) Computational Geometry: Algorithms and Applications, pp. 1–17. Berlin (2008). https://doi.org/10.1007/978-3-540-77974-2_1
8. Eto, Y., Suzuki, M.: Mathematical formula recognition using virtual link network. In: International Conference on Document Analysis and Recognition (ICDAR), pp. 762–767 (2001). https://doi.org/10.1109/ICDAR.2001.953891
9. Filippov, I.V., Nicklaus, M.C.: Optical structure recognition software to recover chemical information: OSRA, an open source solution. J. Chem. Inf. Model. **49**(3), 740–743 (2009). https://doi.org/10.1021/ci800067r
10. Han, K., Xiao, A., Wu, E., Guo, J., Xu, C., Wang, Y.: Transformer in transformer. In: Ranzato, M., Beygelzimer, A., Dauphin, Y., Liang, P., Vaughan, J.W. (eds.) Advances in Neural Information Processing Systems (NeurIPS), pp. 15908–15919 (2021). https://proceedings.neurips.cc/paper/2021/file/854d9fca60b4bd07f9bb215d59ef5561-Paper.pdf
11. He, K., Zhang, X., Ren, S., Sun, J.: Spatial pyramid pooling in deep convolutional networks for visual recognition. Trans. Pattern Anal. Mach. Intell. **37**(9), 1904–1916 (2015). https://doi.org/10.1109/TPAMI.2015.2389824
12. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: Conference on Computer Vision and Pattern Recognition (CVPR), pp. 770–778 (2016). https://doi.org/10.1109/CVPR.2016.90
13. Heller, S., McNaught, A., Stein, S., Tchekhovskoi, D., Pletnev, I.: InChI—the worldwide chemical structure identifier standard. J. Cheminform. **5**(1), 7 (2013). https://doi.org/10.1186/1758-2946-5-7
14. Heller, S.R., McNaught, A., Pletnev, I., Stein, S., Tchekhovskoi, D.: InChI, the IUPAC International Chemical Identifier. J. Cheminform. **7**(1), 23 (2015). https://doi.org/10.1186/s13321-015-0068-4
15. Hu, J., Shen, L., Sun, G.: Squeeze-and-excitation networks. In: Conference on Computer Vision and Pattern Recognition (CVPR), pp. 7132–7141 (2018). https://doi.org/10.1109/CVPR.2018.00745
16. Hu, L., Zanibbi, R.: Line-of-sight stroke graphs and Parzen shape context features for handwritten math formula representation and symbol segmentation. In: International Conference on Frontiers in Handwriting Recognition (ICFHR), pp. 180–186 (2016). https://doi.org/10.1109/ICFHR.2016.0044
17. Ibison, P., Jacquot, M., Kam, F., Neville, A.G., Simpson, R.W., Tonnelier, C., Venczel, T., Johnson, A.P.: Chemical literature data extraction: the CLiDE project. J. Chem. Inf. Comput. Sci. **33**(3), 338–344 (1993). https://doi.org/10.1021/ci00013a010
18. Krenn, M., Häse, F., Nigam, A., Friederich, P., Aspuru-Guzik, A.: Self-referencing Embedded Strings (SELFIES): a 100% robust

molecular string representation. Mach. Learn. Sci. Technol. **1**(4), 045024 (2020). https://doi.org/10.1088/2632-2153/aba947
19. Li, Y., Chen, G., Li, X.: Automated recognition of chemical molecule images based on an improved TNT model. Appl. Sci. **12**(2), 680 (2022). https://doi.org/10.3390/app12020680
20. Mahdavi, M., Condon, M., Davila, K., Zanibbi, R.: LPGA: Line-of-sight Parsing with Graph-based Attention for math formula recognition. In: International Conference on Document Analysis and Recognition (ICDAR), pp. 647–654 (2019). https://doi.org/10.1109/ICDAR.2019.00109
21. Matsakis, N.E.: Recognition of handwritten mathematical expressions. Master's Thesis, Massachusetts Institute of Technology (1999)
22. McDaniel, J.R., Balmuth, J.R.: Kekule: OCR-Optical Chemical (structure) Recognition. J. Chem. Inf. Comput. Sci. **32**(4), 373–378 (1992). https://doi.org/10.1021/ci00008a018
23. Morin, L., Danelljan, M., Agea, M.I., Nassar, A., Weber, V., Meijer, I., Staar, P., Yu, F.: MolGrapher: graph-based visual recognition of chemical structures (2023). https://doi.org/10.48550/arXiv.2308.12234
24. Mouchère, H., Zanibbi, R., Garain, U., Viard-Gaudin, C.: Advancing the state of the art for handwritten math recognition: the CROHME competitions, 2011–2014. Int. J. Doc. Anal. Recognit. **19**(2), 173–189 (2016). https://doi.org/10.1007/s10032-016-0263-5
25. Mouchère, H., Viard-Gaudin, C., Zanibbi, R., Garain, U., Kim, D.H., Kim, J.H.: ICDAR 2013 CROHME: third international competition on recognition of online handwritten mathematical expressions. In: International Conference on Document Analysis and Recognition (ICDAR), pp. 1428–1432 (2013). https://doi.org/10.1109/ICDAR.2013.288
26. Nguyen, A., Huang, Y.C., Tremouilhac, P., Jung, N., Bräse, S.: CHEMSCANNER: extraction and re-use(ability) of chemical information from common scientific documents containing ChemDraw files. J. Cheminform. **11**, 77 (2019). https://doi.org/10.1186/s13321-019-0400-5
27. O'Boyle, N., Dalke, A.: DeepSMILES: an adaptation of SMILES for use in machine-learning of chemical structures. ChemRxiv, pp. 1–9 (2018). https://doi.org/10.26434/chemrxiv.7097960
28. O'Boyle, N.M., Banck, M., James, C.A., Morley, C., Vandermeersch, T., Hutchison, G.R.: Open Babel: an open chemical toolbox. J. Cheminform. **3**(1), 33 (2011). https://doi.org/10.1186/1758-2946-3-33
29. Piroi, F., Lupu, M., Hanbury, A., Sexton, A., Magdy, W., Filippov, I.: CLEF-IP 2012: retrieval experiments in the intellectual property domain. In: Forner, P., Karlgren, J., Womser-Hacker, C. (eds.) CLEF 2012 Evaluation Labs and Workshop. CEUR Workshop Proceedings (CEUR-WS.org) (2012)
30. Qian, Y., Guo, J., Tu, Z., Li, Z., Coley, C.W., Barzilay, R.: MolScribe: robust molecular structure recognition with image-to-graph generation. J. Chem. Inf. Model. **63**(7), 1925–1934 (2023). https://doi.org/10.1021/acs.jcim.2c01480
31. Rajan, K., Brinkhaus, H.O., Zielesny, A., Steinbeck, C.: A review of optical chemical structure recognition tools. J. Cheminform. **12**(1), 60 (2020). https://doi.org/10.1186/s13321-020-00465-0
32. Rajan, K., Zielesny, A., Steinbeck, C.: DECIMER: towards deep learning for chemical image recognition. J. Cheminform. **12**(1), 1–9 (2020). https://doi.org/10.1186/s13321-020-00469-w
33. Ray, L.C., Kirsch, R.A.: Finding chemical records by digital computers. Science **126**(3278), 814–819 (1957). https://doi.org/10.1126/science.126.3278.814
34. Ronneberger, O., Fischer, P., Brox, T.: U-Net: convolutional networks for biomedical image segmentation. In: Navab, N., Hornegger, J., Wells, W.M., Frangi, A.F. (eds.) Medical Image Computing and Computer-Assisted Intervention (MICCAI), pp. 234–241 (2015)
35. Sadawi, N.M., Sexton, A.P., Sorge, V.: Performance of MolRec at TREC 2011 overview and analysis of results. In: Voorhees, E.M., Buckland, L.P. (eds.) Text REtrieval Conference (TREC). NIST Special Publication, vol. 500-296 (2011). http://trec.nist.gov/pubs/trec20/papers/UoB.chem.update.pdf
36. Sadawi, N.M., Sexton, A.P., Sorge, V.: Molrec at CLEF 2012—overview and analysis of results. In: Forner, P., Karlgren, J., Womser-Hacker, C. (eds.) CLEF 2012 Evaluation Labs and Workshop. CEUR Workshop Proceedings (CEUR-WS.org), vol. 1178 (2012). https://ceur-ws.org/Vol-1178/CLEF2012wn-CLEFIP-SadawiEt2012.pdf
37. Schulz, K.U., Mihov, S.: Fast string correction with Levenshtein automata. Int. J. Doc. Anal. Recognit. **5**(1), 67–85 (2002). https://doi.org/10.1007/s10032-002-0082-8
38. Shah, A.K., Dey, A., Zanibbi, R.: A math formula extraction and evaluation framework for pdf documents. In: International Conference on Document Analysis and Recognition (ICDAR), pp. 19–34 (2021)
39. Shah, A.K., Zanibbi, R.: Line-of-sight with graph attention parser (LGAP) for math formulas. In: International Conference on Document Analysis and Recognition (ICDAR), pp. 401–419 (2023). https://doi.org/10.1007/978-3-031-41734-4_25
40. Staker, J., Marshall, K., Abel, R., McQuaw, C.M.: Molecular structure extraction from documents using deep learning. J. Chem. Inf. Model. **59**(3), 1017–1029 (2019). https://doi.org/10.1021/acs.jcim.8b00669
41. Sun, P., Lyu, X., Li, X., Wang, B., Yi, X., Tang, Z.: Understanding Markush structures in chemistry documents with deep learning. In: International Conference on Bioinformatics and Biomedicine (BIBM), pp. 1126–1129 (2019). https://doi.org/10.1109/BIBM.2018.8621264
42. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, L.U., Polosukhin, I.: Attention is all you need. In: Advances in Neural Information Processing Systems (NeurIPS), pp. 5998–6008 (2017). https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf
43. Wang, C.Y., Bochkovskiy, A., Liao, H.Y.M.: Scaled-YOLOv4: scaling cross stage partial network. In: Conference on Computer Vision and Pattern Recognition (CVPR), pp. 13024–13033 (2021). https://doi.org/10.1109/CVPR46437.2021.01283
44. Wang, Y., Zhang, R., Zhang, S., Guo, L., Zhou, Q., Zhao, B., Mo, X., Yang, Q., Huang, Y., Li, K., Fan, Y., Huang, L., Zhou, F.: OCMR: a comprehensive framework for optical chemical molecular recognition. Comput. Biol. Med. (2023). https://doi.org/10.1016/j.compbiomed.2023.107187
45. Weininger, D.: SMILES, a chemical language and information system: introduction to methodology and encoding rules. J. Chem. Inf. Comput. Sci. **28**(1), 31–36 (1988). https://doi.org/10.1021/ci00057a005
46. Xie, S., Girshick, R., Dollar, P., Tu, Z., He, K.: Aggregated residual transformations for deep neural networks. In: Conference on Computer Vision and Pattern Recognition (CVPR), pp. 5987–5995 (2017). https://doi.org/10.1109/CVPR.2017.634
47. Xu, Z., Li, J., Yang, Z., Li, S., Li, H.: SwinOCSR: end-to-end optical chemical structure recognition using a Swin transformer. J. Cheminform. **14**(1), 41 (2022). https://doi.org/10.1186/s13321-022-00624-5
48. Yoo, S., Kwon, O., Lee, H.: Image-to-graph transformers for chemical structure recognition. In: International Conference on Acoustics, Speech and Signal Processing (ICASSP), pp. 3393–3397 (2022). https://doi.org/10.1109/ICASSP43922.2022.9746088
49. Zanibbi, R., Blostein, D., Cordy, J.: Recognizing mathematical expressions using tree transformation. Trans. Pattern Anal. Mach. Intell. **24**(11), 1455–1467 (2002). https://doi.org/10.1109/TPAMI.2002.1046157

50. Zanibbi, R., Pillay, A., Mouchere, H., Viard-Gaudin, C., Blostein, D.: Stroke-based performance metrics for handwritten mathematical expressions. In: International Conference on Document Analysis and Recognition (ICDAR), pp. 334–338 (2011). https://doi.org/10.1109/ICDAR.2011.75