Article

Flowmind2Digital: The First End-to-End Flowmind Recognition and Conversion Approach

Huanyu Liu [1,†], Jianfeng Cai [1,†], Tingjia Zhang[1,†], Hongsheng Li[2], Siyuan Wang[2], Guangming Zhu[2], Syed Afaq Ali Shah[3], Mohammed Bennamoun[4] and Liang Zhang[2,*]

1 School of Artificial Intelligence, Xidian University; {hyliu_5, jfcai_1, tjzhang_1129}@stu.xidian.edu.cn
2 School of Computer Science and Technology, Xidian University; {hsli, siyuanwang}@stu.xidian.edu.cn, gmzhu@xidian.edu.cn
3 School of Science and core member of Centre for AI and Machine Learning, ECU; afaq.shah@ecu.edu.au
4 School of Physics, Maths and Computing, Computer Science and Software Engineering, UWA; mohammed.bennamoun@uwa.edu.au
* Correspondence: liangzhang@xidian.edu.cn
† These authors contributed equally to this work.

Flowcharts and mind maps, collectively referred to as flowmind, play an important role in our daily, and many companies have developed dedicated tools. Hand-drawn flowminds offer significant advantages for real-time and collaborative communication. However, there is an increasing demand to convert them into a digital format for further processing. Automated conversion methods are crucial in addressing the challenges associated with manual conversion, such as the cost of time and learning. Previous works have proposed diverse sketch recognition methods. However, these methods face significant limitations in practical situations. Firstly, most methods are designed for specific fields, making it challenging to extend their usage to other fields. Additionally, none of these methods address the critical step of digital conversion after recognition, which is essential to users. Moreover, existing datasets exhibit significant biases relative to actual data, hindering the methods' generalization ability.

Our paper proposes the *Flowmind2digital* method and *hdFlowmind* dataset to address the aforementioned challenges. *Flowmind2digital* is the first comprehensive recognition and conversion method for flowminds, utilizing a neural network architecture and keypoint detection technology to enhance overall recognition accuracy. Our *hdFlowmind* dataset consists of 1,776 hand-drawn and manually annotated flowminds, covering 22 scenarios and surpassing existing datasets in size. Our experiments showcase the effectiveness of our method, with an accuracy rate of 87.3% on the *hdFlowmind* dataset, surpassing the previous state-of-the-art work by 11.9%. Additionally, our dataset demonstrates effectiveness, with a 2.9% increase in accuracy after pre-training and fine-tuning on Handwritten-diagram-dataset. We also highlight the importance of simple graphics for sketch recognition, which can improve accuracy by 9.3%.

Citation: Huanyu Liu, Jianfeng Cai, Tingjia Zhang, Hongsheng Li, Siyuan Wang, Guangming Zhu, Syed Afaq Ali Shah, Mohammed Bennamoun, and Liang Zhang, Flowmind2Digital: The First End-to-End Flowmind Recognition and Conversion Approach. *Electronics* **2024**, *1*, 0. https://doi.org/

Received:
Revised:
Accepted:
Published:

1. Introduction

Sketching is a prevalent and innate communication skill in human society, dating back to ancient times and evident in the spontaneous drawings of infants. There are two practical forms of sketches: (a) flowcharts, which visually represent the sequence of steps in a process and serve as an indispensable tool for documenting and revealing one's thought process, and (b) mind maps, which are visual thinking tools that enable us to structure our ideas and create an intuitive framework around a central concept. Due to their usefulness, many companies have developed software for creating these sketches, collectively referred to as flowmind. However, flowminds are typically not created in any specific digital format but rather hand-drawn by users. These hand-drawn flowminds can be created during brainstorming and planning at a meeting, note-taking for academic research, or process planning as shown in Fig. 1. Hand-drawn flowminds have the advantage of real-time notation, as they only require a pen and any available background to immediately record structural information. However, interacting with software is not as direct, causing inconvenience in many scenarios.

Figure 1. Flowminds used in Various Scenarios

However, there is a need to eventually convert these hand-drawn flowminds into digital format for the sake of clarity, better documentation, and record keeping. The process of converting hand-drawn flowminds into digital format involves a large amount of manual work on the software, including dragging each shape to the right position and typing textual labels for connectors (such as lines, arrows, and double arrows). This task requires considerable time and effort, especially for normal users who are not proficient in using such software.

To alleviate the burden of manual conversion, there is a need for an automated method that allows regular users to obtain digital flowminds directly on commonly used software from their hand-drawn input. This can be achieved in two steps: first, recognizing the elements in a hand-drawn flowmind; and second, converting the recognized elements into common formats using certain digital tools.

Existing methods have limitations and assumptions when it comes to recognizing elements in a hand-drawn flowmind, and they often produce output in a format that is not commonly used. For example, the UML model proposed by [1] only allows users to draw in predefined formats, which does not align with the initial sketch's intention. HDBPMN, proposed by [2][3], offers a comprehensive approach that generates an XML file of a BPMN model from sketch input. However, this format is not commonly used in daily scenarios. Furthermore, no existing method has met the requirement for the final conversion to a commonly used format.

Existing datasets have limitations in terms of real-life scenarios, as they mainly consist of samples drawn on electronic devices. For instance, the Handwritten-diagram-dataset by [3] lacks diversity in terms of input sources. In contrast, real-life scenarios include photos of whiteboards or glass boards, which can present challenges such as overexposure or reflections, and are common sources of initial input. Therefore, there is a need for a comprehensive recognition and conversion method that can interface with commonly used software, such as Microsoft Power Point (PPT) and Visio, and a dataset that covers a larger scope of real-world scenarios.

Hence, our work makes two significant contributions. ***Firstly***, we present *Flowmind2digital*, an approach that utilizes a neural network architecture based on the Arrow-RCNN model proposed by [4] to automatically convert hand-drawn flowminds into editable PPT and Visio digital diagrams. We aim to improve the practicality of the existing approaches for normal users. ***Secondly***, we introduce *hdFlowmind* dataset, which consists of 1,776 images and 27,804 annotations. This dataset covers a larger scope of scenarios and is considerably larger than previous datasets. Additionally, we include 485 samples of basic shapes in the dataset to demonstrate their importance in the experiments. We make this dataset publicly available for future research. Our experiments on this dataset indicate that Flowmind2digital outperforms the state-of-the-art models, which validates the effectiveness of hdFlowmind. Lastly, we also provide the first ultra-lightweight version of the Visio-Python kit that facilitates direct programming operations on Visio software.

Our work in this paper significantly extends the scope and quality of the Arrow-RCNN approach, which focuses on flowchart recognition and keypoint detection. Specifically, our contributions are as follows:

- We introduce an improvement based on the human keypoints detection that improves the accuracy of Arrow-RCNN.
- We extend the model to cover the recognition of specific content of text box, which overcomes the limitation of Arrow-RCNN model that only recognizes the location of text.
- We further achieve compatibility with the internal models of Detectron2[1], providing a pre-trained model that demonstrates the usefulness of our *hdFlowmind* pre-training approach for training in other tasks within the relevant sketch domain.

The rest of this paper is organized as follows: We provide an overview of existing methods and related work in the field of sketch recognition in Section 2, followed by several challenges that exist in the recognition scenario we focus on in Section 3. Section 4 provides a detailed description of our *hdFlowmind* dataset. Then we introduce our *Flowmind2digital* recognition model in Section 5 and evaluate our approach and dataset in Section 6. Finally, we discuss the implications and limitations of our approach in Section 7. The paper is concluded in Section 8.

2. Related Works

Drawing sketches is a useful method to convey information quickly and effectively. As a result, several sketch recognition approaches and datasets have been proposed to automate the conversion of sketches to digital formats. Microsoft's Visio and PowerPoint software have become widely used for creating digital diagrams. In this context, our work concentrates on flowmind recognition using keypoint detection and post-processing the results for use in PPT and Visio software. Therefore, we discuss the following related works in this paper: (1) existing flowmind recognition methods and datasets; (2) generic keypoint detection methods; (3) approach to interface with PPT and Visio software.

2.1. Flowmind Recognition

The process of flowmind recognition can be divided into three tasks including 1) shape recognition to locate and classify various basic shapes 2) connector recognition to identify the connector between each shape and the specific connection revealed using keypoints 3) text recognition to locate the text label and identify the specific content.

Conventionally, sketch recognition can be differentiated into online and offline methods which have diverse patterns. The input to online method is mainly a series of sequence strokes on geometry, stroke or gesture base. There are also datasets, including FC_A proposed by Awal[5] , FA and FC_B proposed by Bresler et al.[6], DIDI [7] proposed by Gervais et al. etc. [8][9]

In comparison to the online approaches, offline sketch recognition simply needs raw images instead of sequential strokes, making it more suitable for real world situations. However, this also brings greater complexity, given that less information is contained in the input. Some works [10][11][12] tried to convert it to online problem by reconstructing the strokes of the hand-drawn flowcharts, but it is not applicable in the scenario we handle in this work which contains noises and raw images.

Therefore, the object-based method is also proposed, requiring corresponding datasets to train the models. Since the datasets of online and offline methods overlap, the online dataset can be converted into the offline datasets through certain transformation. Wu et al. [13] proposed to use FC_A dataset to train offline recognition model [5][14]. Bresler extracted two datasets from the FC_B dataset – D_a and D_b [6][15][16] for offline recognition.

[1] https://github.com/facebookresearch/detectron2.git

At the same time, several datasets have been created to support sketch recognition for various scenes, including Bernhard's HDPBPMN dataset for BPMN graphs [3]. However, these datasets have limitations in terms of comprehensiveness, particularly in terms of exposure, drawing tools, and materials. These limitations will be discussed further in Section 3. Table 1 provides an overview of the attributes of these datasets for different sketch scenes.

Table 1. Related Work and Datasets

Name	Category	Authors	Objects			Connection	Non-digital		Digital		DB	DPC	DPT
			Shapes	Connectors	Textboxes	Keypoints	Exposure	Blur	Pen-Pressure	DE			
FC_A[5]	Flowcharts	Awal et al.	Yes	Yes	Yes	Yes	-	-	No	No	No	No	No
FA[15]	Finite automata	Bresler,Pr_u_a	Yes	Yes	Yes	Yes	-	-	No	No	No	No	No
FC_B[6]	Flowcharts	Bresler,Pr_u_a	Yes	Yes	Yes	Yes	-	-	No	No	No	No	No
FC_Bscan[16]	Flowcharts	Bresler,Pr_u_a	Yes	Yes	Yes	Yes	-	-	No	Yes	No	No	No
DIDItext[7]	Flowcharts	Philippe Gervais	Yes	Yes	Yes	Yes	-	-	Yes	Yes	No	Yes	Yes
DIDIno_text[7]	Flowcharts	Philippe Gervais	Yes	Yes	No	Yes	-	-	Yes	Yes	No	Yes	Yes
hdBPMN2021[3]	BPMN models	Bernhard Schafer	Yes	Yes	No	Yes	Yes	Yes	No	No	Yes	Yes	Yes
hdBPMN2022	BPMN models	Bernhard Schafer	Yes	Yes	Yes	Yes	Yes	Yes	No	No	Yes	Yes	Yes
hdFlowmind	Flowcharts, Mind map	Ours	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes

*DB = different backgrounds, DE = different equipment DPC = different pen colors, DPT = different pen thickness

In terms of detection model, Julca-Aguilar and Hirata used the Faster-RCNN model [17] in target detection to identify flowchart elements [18]. However, traditional object detection model can only tell where the arrows in a flowchart is, but not their correspondence (to where and what) . To address such limitation, Bernard et al. proposed Arrow-RCNN [4], a model developed from Faster-RCNN, capable of detecting the head and tail of arrows in flowcharts, but lacked post-processing step to digitalize the result. Bernard et al. also proposed SketchToProcess [19], which realized the direct transformation of hand-drawn BPMN diagrams into corresponding standard BPMN xml files. However, the problem of connection orientation is still not solved. The scope of use is relatively narrow because of the limitations in recognizing the multiple connectors that commonly appear in mind mapping.

It is worth noting that most existing approaches do not provide a way to connect post-processing with previous software to generate editable graphics, which is a crucial requirement in practical applications. Most methods simply recognize and output raw data without further processing. The Sketch2Process model is an exception as it generates BPMN XML files, but even then, additional processing by users is still required. In practice, normal users require software-editable graphics rather than raw data that needs further steps. Therefore, the limitation of existing transformation models is evident from a practical perspective.

In summary, the analysis presented above highlights the active research field of sketch recognition for raw-sketch transformation, which has seen various approaches and datasets being introduced. However, there are still considerable gaps in the existing datasets in terms of their scope and quantity. Moreover, the current approaches do not consider the crucial step of connecting the recognized sketches to relevant software for practical use.

2.2. ***Keypoint Detection***

Recognizing the connections in flowmind is a challenging task as it requires accurately identifying the trajectory of each connection. However, a more flexible approach is to simplify the target by recognizing a limited number of keypoints, which can improve the accuracy and flexibility of subsequent adjustments and generation in software.

Keypoint detection has numerous applications, one of which is human posture estimation (HPE) that includes the detection of facial, hand, and human keypoints. HPE systems such as those presented in [20][21] typically follow a top-down approach for multi-keypoint detection. In the first stage, object detectors determine the location of each instance in

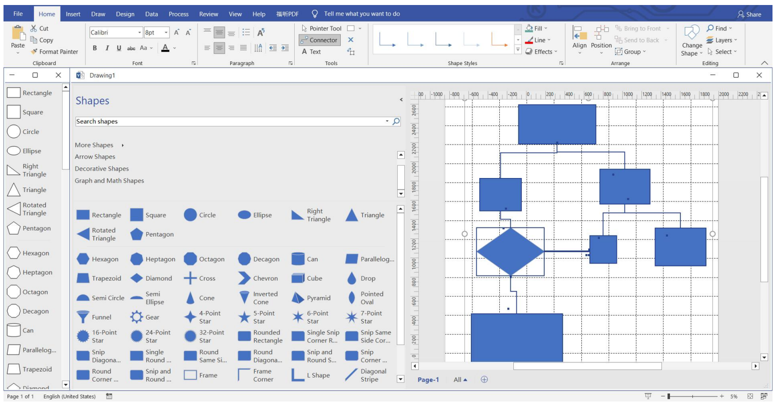


Figure 2. Visio Interface

an image. In the second stage, instances are clipped from the original image, adjusted to a specific resolution, and fed to an HPE network that outputs the keypoints' location. Currently, there are two main methods.

The first approach for keypoint detection involves using the keypoint location as the target for regression and directly outputting the location of the keypoint through a fully connected layer, without generating prediction results for each pixel. While this method has a lower model complexity and can achieve some end-to-end full differential training, it also has some obvious problems. These problems mainly include lower accuracy and generalization due to direct regression, no theoretical error lower bound, and a higher risk of overfitting because the task of keypoints is relatively flexible compared to HPE. Therefore, we do not use this method. The second approach is to estimate the heatmap based on keypoint location, which generates a heat value for each pixel position that indicates the probability of the corresponding pixel being a keypoint. Heatmap-based methods have been evaluated and have been shown to have higher accuracy in recent years, but there are still slight differences in the task of connector keypoints.

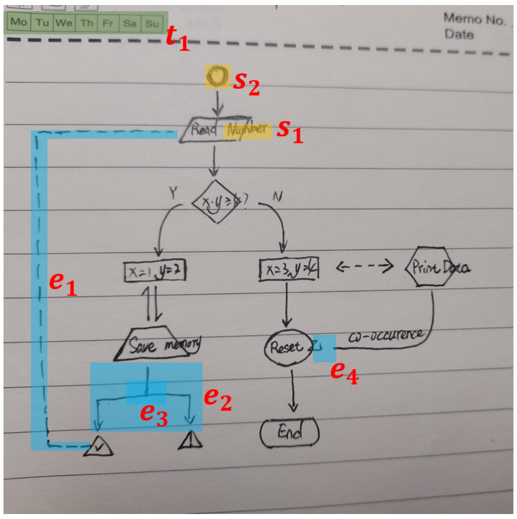
The traditional COCO dataset includes person instances that may be occluded, underexposed, or blurred, resulting in varying levels of visibility for the keypoints. The visibility is usually divided into three levels in general: "0" indicates that there is no keypoint. "1" indicates a invisible keypoint due to occlusion, and "2" indicates an existing visible keypoint. However, unlike keypoint detection for human posture estimation, where keypoints may have different levels of visibility due to occlusion, underexposure, or blur, keypoints of connectors typically have a clear and binary visibility state of either "0" or "2" since the head and tail of almost all arrows are clearly visible. Thus, the original keypoint detection method used for human posture estimation cannot be directly applied to connector keypoint detection. In this paper, we propose a modified model tailored to our specific needs, as discussed in Section 5.

2.3. *Related Software*

PowerPoint, a presentation developed by Microsoft Corporation, is widely used in flowchart creation and mind map editing. It has become a fundamental software tool in many industries. Visio is a graphics software tool in the Microsoft Office toolkit, which helps in understanding complex systems and processes and making better decisions. Visio diagrams can be saved in formats such as .svg and .dwg. As PowerPoint and Visio are widely used, our proposed solution aims to interface with these software tools. Fig. 2 shows an example of the Visio interface.

Currently, the commonly used method for interacting with these software is to use the PPT and Visio API provided by Microsoft's official website[2] [3]. This API offers detailed shape

[2] https://learn.microsoft.com/office/vba/api/overview/visio
[3] https://learn.microsoft.com/office/vba/api/overview/powerpoint



Shape recognition challenge
- s_1 → **Object exceeding the bounding box**
- s_2 → **Shape mussy strokes**

Connector recognition challenge
- e_1 → **Connector go through the entire graph**
- e_2 → **Multiple connection to different objects**
- e_3 → **Connector cross each other**
- e_4 → **Connector not connected to corresponding objects**

Text recognition challenge
- t_1 → **Backgrounds of printing fonts**

Figure 3. Example of a flowmind with various highlighted recognition challenges

interfaces, internal functions, and proprietary properties, but it is primarily intended for developing their own extensions, making it difficult to interface with our model. Therefore, we have opted to use third-party extension packages to establish the interconnection.

The Python extension package "PPTX"[4] is used for interacting with PowerPoint software, and it adheres to the rules of PPT documents. The package defines pages as containers, shapes as objects, and connection lines as entities. Using this package, objects in PowerPoint can be manipulated through the Python language.

Regarding the Visio software, although there are third-party extension packages such as vsdx, their functions are too limited to meet basic needs. Therefore, we developed our API using the third-party extension package win32com provided by Python. This API mainly allows us to call the underlying components of Word, Excel, PPT, Visio, and other software.

3. Challenges

This section discusses the difficulties in recognizing hand-drawn flowminds, which result from various real-world factors that are not adequately represented in existing datasets. We will focus on the challenges of shape recognition, connector recognition, text recognition, and the impact of the raw backgrounds. To illustrate these challenges, we will use the example drawing shown in Fig. ??, which is taken from our *hdFlowmind* dataset.

Shape Recognition Challenge

Recognizing all shapes and their positions in a flowmind is a major challenge in flowmind recognition, as it is the first step in the process. Shapes in a flowmind are referred to as nodes, and can include rectangles, diamonds, arrows, and text, each representing a different node. Shapes are defined by a bounding box, which contains information about the shape's location and type. However, recognizing shapes can be quite complex due to several challenges, including:

1. One challenge in shape recognition is the ambiguity between similar shape types. For example, ellipses and long ellipses have a high degree of similarity, with the curvature of the edge being the only distinguishing factor (curvature is non-zero for ellipses, and zero for long ellipses). However, in practice, people tend to draw them similarly, which can lead to confusion in recognition.
2. The second challenge in shape recognition of hand-drawn flowminds is that the objects often exceed the bounding box of the shape, as shown in problem s1 in Fig. ??.

[4] https://python-pptx.readthedocs.io

This makes it difficult to distinguish between different shape types. Moreover, the boundary of the shape may be drawn multiple times, resulting in messy strokes, as depicted in problem s2 in Fig. ??.

Connector Recognition Challenge

The connectors in flowcharts and mind maps play a crucial role in indicating the relationships between different nodes. Although many datasets classify connectors as a single type, they can actually be differentiated into three graphic styles, namely line, single arrow, and double arrow, which need to be recognized separately.

Recognizing connectors is more complex than recognizing shapes since connecting lines not only need to be identified in terms of their type and location, but also require the determination of the shapes they connect and the relationship between them. Some of the challenges associated with connector recognition are:

1. The flexibility of the connection line path in hand-drawn flowminds can cause connectors to go through the entire graph, resulting in the bounding box including a large portion of other objects (e1 in Fig. ??).
2. Many-to-Many connections occur when a node extends multiple connection lines to different objects (e2), resulting in a high intersection over union (IOU) between the bounding boxes of these objects.
3. The crossing and intersection of connection lines, as shown in e3, can complicate the identification process since lines often cross over or intersect with other objects.
4. In rough sketches, bad connections are common where the drawn connectors are not always connected correctly to the corresponding object (e4). This issue makes it difficult to identify, especially when multiple possible objects are involved.

Text Recognition Challenge

Recognition of text involves identifying both the textboxes and their respective positions, as well as the specific content within them. This task presents a few challenges such as:

1. Handwriting recognition (HWR) is required to recognize the specific text content in the text box after locating it. However, in hand-drawn flowminds, the text often has more background interference compared to general HWR. For instance, people may draw their sketch on any background according to their convenience, and some backgrounds may even include noises such as printed fonts (t1).
2. Text that goes beyond the boundaries of a text box and intersects with other shapes or connectors is referred to as "Text out of text box". This can occur when the content of the text is too long and goes beyond the bounding box. Detecting this type of text presents similar challenges to those encountered when detecting shapes and connectors.

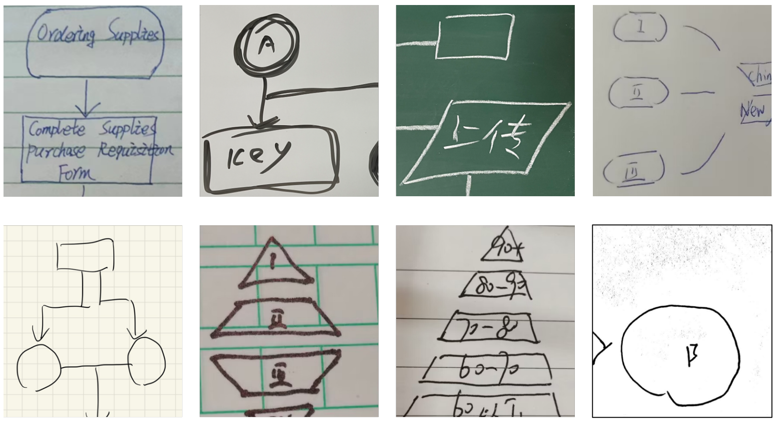


Figure 4. Various Backgrounds in Flowminds

Other Challenges

Apart from the challenges mentioned earlier, there may be additional external factors that could impact the recognition process. The challenges in this case include:

1. The background of a hand-drawn flowmind can vary depending on the type of paper used, which may include horizontal lines, grids, squares, dashed lines, or blank spaces. The additional lines in the background may resemble the flowmind shapes or connectors drawn by the user, which can cause confusion during identification. Other scenarios may involve drawing on whiteboards, blackboards, or glass boards. Additionally, some users may use electronic devices for sketching, such as tablets or scanning software, which can result in variations in the background. Examples are shown in Fig. 4.
2. The quality of the raw sketch can be influenced by the variability in drawing tools, such as the clarity, consistency, and thickness of the lines drawn.
3. The equipment used to generate the input image can vary. Screenshots or scans taken directly on electronic devices typically have high clarity. However, taking a photo of a raw sketch on paper can result in rotated, blurry images or incomplete content. Examples illustrating this are shown in Fig. 5.
4. Another challenge lies in the post-processing stage after recognition. Existing methods do not consider the interface with software, leaving a gap in this aspect. We believe that we are the first to propose an interface with specific software for generating editable graphics

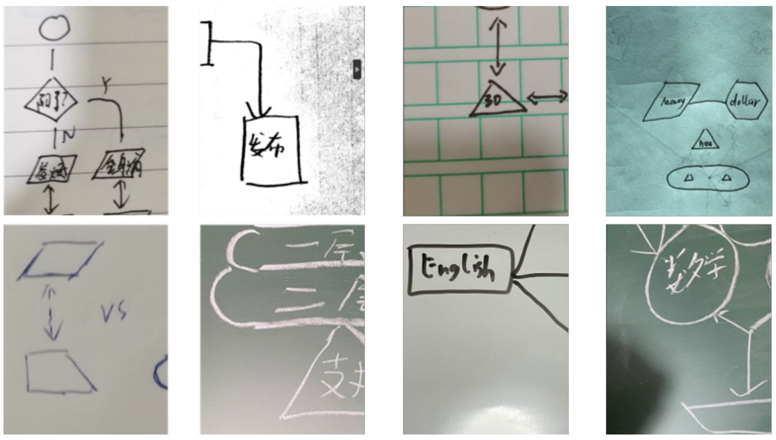


Figure 5. Recognition Challenges Resulting from Different Methods of Digitizing Flowminds

4. Dataset

This section discusses the Collection, Annotation, Characteristics and Splitting of the *hdFlowmind* dataset, that is publicly available [5]. Furthermore, emphasis is placed on the advantages of our dataset over relevant datasets in terms of quantity, quality, and diversity, showed in Table 1. This emphasis aims to comprehensively address and overcome the various challenges mentioned in the Section 3.

4.1. Dataset Collection

Our dataset comprises 1,776 hand-drawn flowminds collected from XIDIAN University. These images include hand-drawn flowcharts found on the Internet, mind maps used in practical applications, as well as sketches made during meetings or brainstorming sessions. The participants were instructed to ensure that their designs and strokes reflect real-life scenarios.

[5] https://huggingface.co/datasets/caijanfeng/hdflowmind

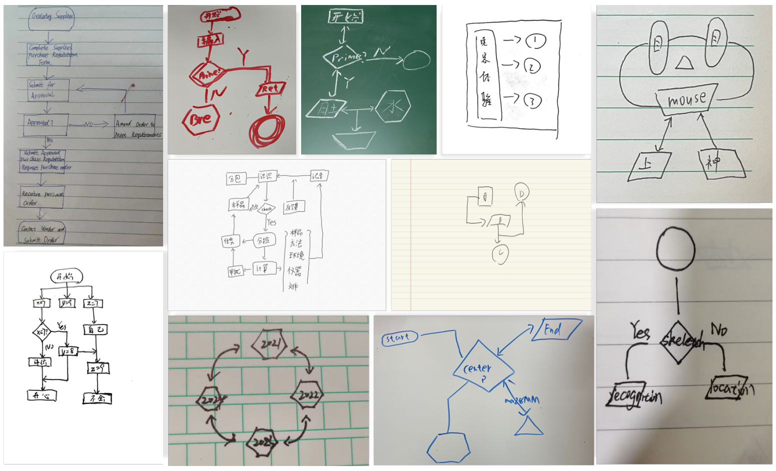


Figure 6. Overview of the *hdFlowmind* Dataset

To cover a larger scope of scenarios, we asked participants to draw with different drawing mediums, backgrounds, and photographic equipment. This intentional diversification in the dataset creation process ensures a more comprehensive representation of real-world conditions and user behaviors. Our dataset encompasses a rich variety of hand-drawn sketches captured under diverse conditions, including varying lighting, textures, and drawing styles. Unlike some existing datasets that may have limited diversity in terms of drawing styles or environmental conditions mentioned in Section 2, our dataset embraces a broader spectrum of user interactions and artistic expressions. This diversity in the data collection process enhances the generalization capability of our model, making it well-suited for real-world applications where visual inputs can vary significantly.

In the category of non-digital samples, We have a variety of non-digital samples in our dataset, which were drawn on different backgrounds such as whiteboards, glass boards, standard A4 paper, brown and yellow paper, thin paper (which is translucent and the content on the reverse side can be seen), grid paper, quadrille paper, lined paper (with line color in green or black), ruled paper, and chart paper (which has some patterns and text on it). The drawing tools used in these samples include markers, black pens, blue pens, whiteboard pens, chalks, and more. Additionally, the circumstantial conditions under which these samples were drawn include normal lighting, dusky lighting, overexposure, and shadow occlusion. We also have some samples that were scanned using software[6].

For digital samples, participants used electronic devices such as Apple iPad and Samsung Pad, along with electronic pens, to create the sketches. They used two different software programs, Concept drawing board and Notability[7] [8]. The backgrounds for these sketches included grids, lattices, horizontal lines, and scattered dots in various colors. The pen settings varied, including brush strokes with pressure sensing, fix jitter pen, and soft pencil tools provided in the software.

The overview of *hdFlowmind* is provided in Fig 6.

4.2. *Dataset Annotation*

We utilized the PASCAL VOC format, which is a widely used format for Object Detection datasets in the field of Computer Vision, to annotate the shapes and connections in each sample for both training and evaluation purposes. To generate the annotation files, we utilized a data annotation tool provided by Huawei Cloud[9]. These annotation files

[6] https://www.camscanner.com/
[7] https://concepts.app/
[8] https://notability.com/
[9] https://cloud.huawei.com

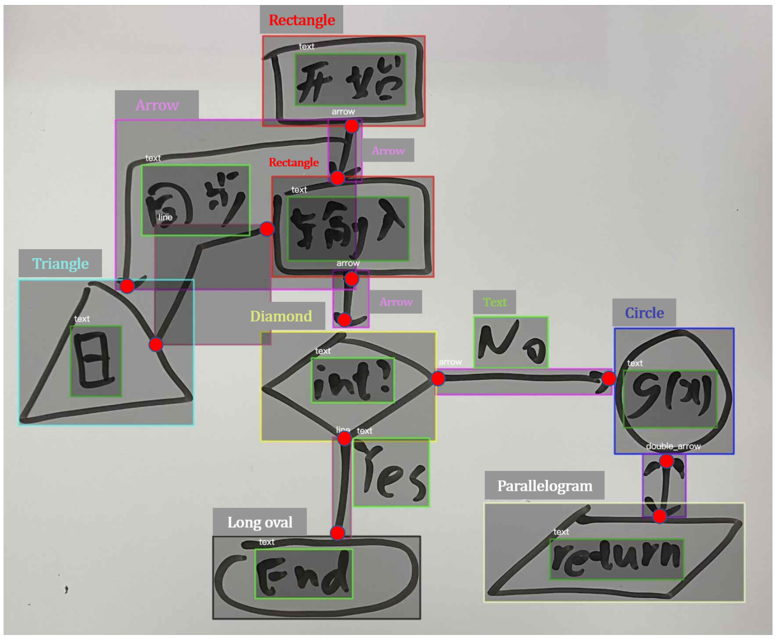


Figure 7. An Example Image with Annotations

consist of labels, four coordinates of bounding boxes, and additional coordinates of head and tail for connectors, which include arrow, double arrow, and line.

Please note that the annotation tool we used doesn't allow distinguishing whether keypoints are necessary for each instance. Hence, we used the coordinates of keypoints as attribute values of bounding boxes during annotation, and manually entered the coordinates afterward. After careful checking, the error rate was no more than 0.225% (4/1776). Fig. 7 shows a sample from our *hdFlowmind* dataset.

4.3. *Dataset Characteristics*

Our *hdFlowmind* dataset comprises 1,776 images, with 17,652 annotated bounding boxes and 10,152 keypoints. Table 2 illustrates that the dataset covers 12 flowmind elements, including 7 basic shapes, 1 text box, and 3 types of connectors. The minimum number of annotations among hdFlowmind images is 3, while the maximum is 93, with an average annotation count of 22.56 per image. On average, each image contains 14.23 elements. Compared to similar datasets, hdFlowmind includes a larger variety of compositions and scenarios, with the most extensive quantity of images and annotations.

The challenges in recognizing these images are mainly related to the issues discussed in Section 3, such as different backgrounds (paper, board, electronic devices, etc.) and drawing tools (markers, chalk, pens, etc.), as well as image-capturing problems (blurry images, exposure, etc.). Additionally, the recognition of shapes, connectors, and text also poses significant difficulties. These challenges are comprehensively presented in Table 3, while Fig. 8 displays various types of shapes used in developing our dataset.

Our publicly available flowmind dataset has a wide range of elements and a high degree of composition diversity, which makes it valuable for research and development purposes. Additionally, our elements and compositions are practical and natural.

4.4. *Dataset Splitting*

To follow the protocol of related hand-drawn diagram datasets [6][16][15], we split the *hdFlowmind* dataset into three parts: training, validation, and testing. In comparison to professional field datasets, such as BPMN dataset [19], our flowmind dataset exhibits higher variability in terms of composition, backgrounds, and drawing medium. This implies that different flowminds from the same participant differ significantly. Thus, we randomly

Table 2. *hdFlowmind* elements in the 1,776 annotated images

Element	Name	Count
Basic shapes	circle	1,039
	diamonds	599
	hexagon	562
	long oval	485
	parallelogram	700
	rectangle	2,209
	trapezoid	460
	triangle	602
Text	textblock	5,920
Connectors	arrow	3,219
	double arrow	634
	line	1,223
Maximum of annotations per image: 93		
Minimum of annotations per image: 3		
Average of annotations per image: 22.56		
Average of element per image: 14.23		

Table 3. Statistics of challenges posed by different external features in our dataset

Group	type	count
background	whiteboard	54
	blackboard	48
	grid paper	224
	ruled paper	171
	brown paper	200
	white paper	155
	digital	638
	single	485
drawing tools	marker	54
	chalk	48
	red pen	4
	blue pen	31
	black pen	681
	electronic pen	1,123
Other	motion blur	158
	exposure	7

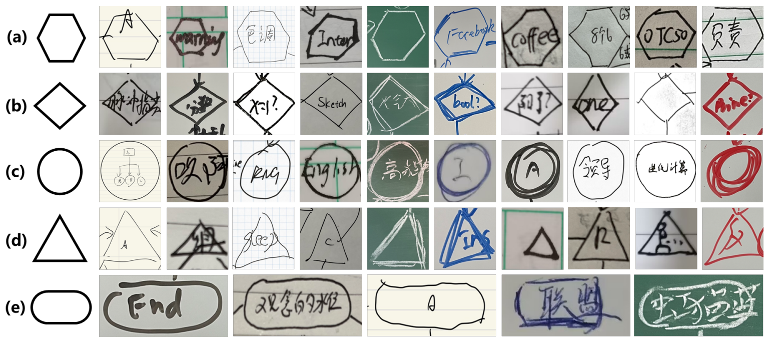


Figure 8. Examples of different hand-drawn shapes

Figure 9. Examples of Basic Geometric Shapes

divided the dataset into three parts with a 6:2:2 ratio, i.e., the training set contains 775 samples, the validation set has 258 samples, and the test set has 258 samples.

f, to enhance the recognition accuracy of basic shapes with less frequency, we included an additional training image containing 107 parallelograms, 94 hexagons, 93 rhombuses, 102 trapezoids, 89 triangles, and a total of 485 basic images as depicted in Fig. 9. By adding this extra image, the ratio of train/validation/test sets became 1260:258:258. We assessed the impact of these auxiliary images on the training results in Section 6.

5. Methods

This section describes the *Flowmind2digital* method for creating digital flowmind diagrams on PPT and Visio from hand-drawn inputs. As shown in Fig. ??, *Flowmind2digital* consists of two main components: object & keypoint detection, and post-processing.

Keypoint detection

5.1. *Object & Keypoint Detection*

Mask-RCNN

The Mask-RCNN is a neural network approach based on the Faster-RCNN and is known for its high accuracy and extensibility, as described by [21]. This approach can also be used for the keypoint detection, as demonstrated in the example of human posture recognition. The two-stage architecture, inherited from Faster-RCNN, provides a robust framework for effectively handling object detection tasks. This accuracy is crucial for tasks, where precise localization of keypoints is paramount.

The initial input of Mask-RCNN is an RGB image represented as a three-dimensional array. The first two dimensions correspond to the size of the image, while the third

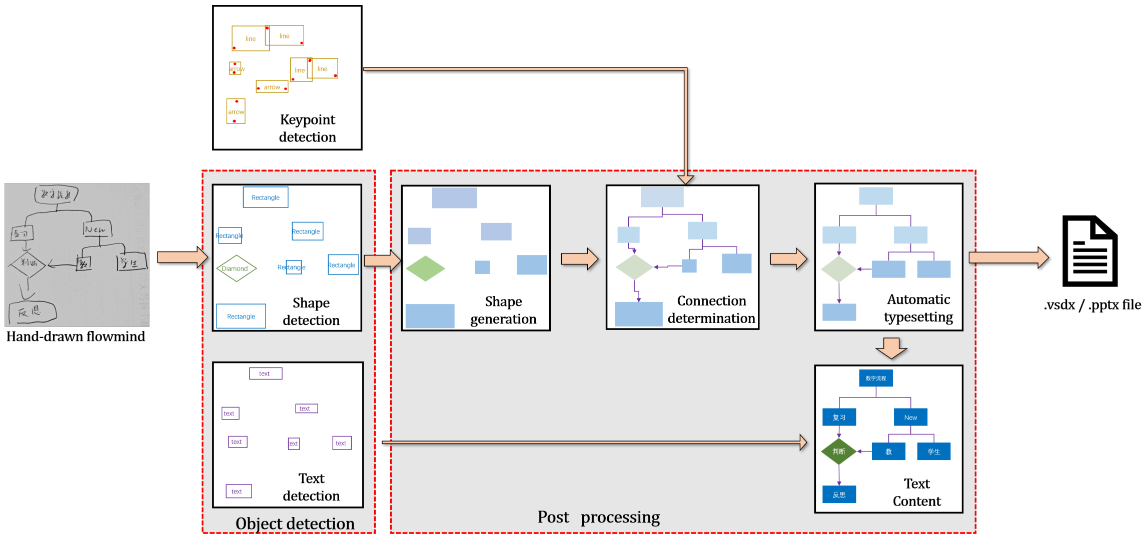


Figure 10. Overview of our approach: Given a handwritten image, we first perform object detection for shapes, keypoints and text. Next, we generate the shape and connector based on coordinates and connection relationships in the relevant software. Then, we adopt a clustering-based automatic layout algorithm and fill in the corresponding text boxes detected by optical character recognition. Finally, we generate a visual file of the software.

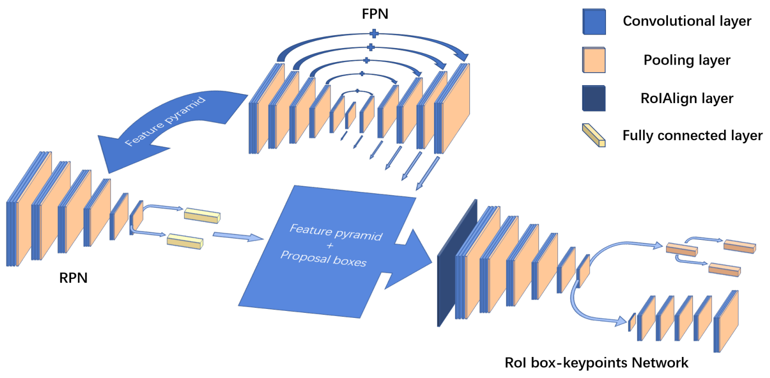


Figure 11. Schematic of the Recognition Network in Our Model

dimension represents the three color channels: red, blue, and green. The detection process is based on the feature map of the image, which is learned by the backbone network, a FPN network in our application. Multi-scale feature is extracted and fused by up-sampling to combine the semantic and visual information.

The Mask-RCNN follows a two-stage process, similar to Faster-RCNN, for object detection. In the first stage, the Region Proposal Network (RPN) generates region proposals. Each proposal represents a bounding box around a region of interest along with a confidence score that indicates whether it belongs to the foreground or background. The second stage of Mask-RCNN involves a ROI (Region of Interest) network that classifies each proposal and refines the bounding box location. The ROI Align mechanism is used to extract a fixed regional feature map that corresponds precisely to the proposal region. This smaller feature map is used to predict refined bounding boxes and a score distribution over defined classes for each foreground proposal. Finally the Mask-RCNN gives an output set of object (x, c, s), where x represents coordinates, c represents the most likely class and s represents the confidence score.

Moreover, the ability of Mask-RCNN to perform keypoint detection aligns well with the specific requirements of the application. The mechanism for region proposal generation in the first stage, coupled with the ROI network in the second stage, enables not only accurate bounding box localization but also detailed keypoint predictions.

According to[22][23], we will conduct a comprehensive analysis of trade-offs in the entire system pipeline, considering aspects such as program runtime, memory usage, accuracy, speed, and correctness. Detailed experiments are presented in Section 6 to provide a thorough examination of the rationality behind the utilization of Mask RCNN.

Trade-off in Speed and Time Complexity

While Mask-RCNN may not be the fastest algorithm, the trade-off in speed is justified by the enhanced accuracy achieved through the two-stage process. The use of the Region of Interest (ROI) Align mechanism contributes to a time complexity suitable for our application. Comparatively, YOLO[24] may offer faster inference times but at the expense of potential sacrifices in accuracy.

Memory Efficiency and Multi-scale Feature Fusion

The FPN backbone utilized in Mask-RCNN contributes to effective memory utilization, accommodating the processing of high-resolution images. The multi-scale feature extraction and fusion through up-sampling enhance the model's ability to capture both semantic and visual information, a feature not explicitly highlighted in some other architectures. DETR[25], while novel in its transformer-based architecture, may have different time complexity considerations as it directly predicts object bounding boxes and classes in a single pass. Due to the parallelization mechanism of multi-heads in the DETR transformer architecture being optimized for GPUs, it cannot run efficiently on lightweight CPUs.

Over all, based on previous research work and analysis above, we utilize the Mask-RCNN for our object detection network.

Keypoint detection is a popular extension of the Mask-RCNN model. During the second stage, the ROI network is altered to extract keypoints and refine bounding boxes at the same time. The ROI Box-Keypoints network involves two parallel fully connected layers, which are applied after the feature map array is transformed. One of the layers is responsible for refining the bounding box by regression, while the other layer treats keypoint detection as a pixel-level classification problem. Each arrow instance is assigned a one-hot mask, with the keypoint pixel defined as foreground and the rest as background.

We utilized a pre-trained FPN-resnet50 model, which was originally developed for human posture recognition. The selection of FPN-resnet50 as the backbone is motivated by its well-established balance between high accuracy and robustness. Additionally, it facilitates seamless experimental comparisons with other models of a similar class. But we customized it for our purpose. **Specifically, we set the number of keypoints to be detected as two.** This was the basic number needed to ensure that the detector can identify the connection between two shapes. The path of the connector can be easily adjusted through automatic typesetting or manual post-processing, that we need no additional keypoints to indicate the specific path of the connector. Therefore, two keypoints are sufficient to capture the composition of the flowmind diagram.

5.2. *Post processing*

After obtaining the results of identifying key points for objects and connectors as described earlier, our attention turns to the human-computer interaction process. We aim to create a technique that enables the conversion of these results into a digital format. This involves generating files such as .pptx or .vsdx from the output.

Our approach to accomplishing this task involves the creation of a post-processing procedure, as described in Section 2. This procedure can be broken down into four distinct parts.

Firstly, the shape generation component uses Visio and PPT to create the corresponding shapes at the coordinates detected during keypoint recognition.

Secondly, the connection determination component identifies the exact point on the shape where the detected connector should connect, based on the keypoint information.

Thirdly, the text content component assigns labels to the shapes and connectors, generates text boxes, and extracts the relevant content using OCR software.

Lastly, the automatic typesetting component adjusts the sizes and positions of the shapes based on intelligent clustering, and generates the final output.

Shape generation

As discussed in Section 2, we utilize the python-pptx library to establish communication with Microsoft Powerpoint. This library provides Python classes to interact with the elements present in a PPT document such as slides, shapes, and connectors. In the case of Visio, since there is no suitable toolkit available, we use the win32com[10] library to interact with the program. The Visio template is read and placed onto the created page object.

As mentioned in Section 2, we use the python-pptx library to interface with Microsoft Powerpoint. The elements of slides, shapes or connectors in ppt document is operated as python classes. Since there's no proper toolkit to interface with Visio, we use the win32com to operate it with this program. The template in Visio is read and placed onto the created page object. To ensure proper formatting, we need to convert the coordinates of the bounding box (x_0, y_0, x_1, y_1) to the format of (x_c, y_c, H, W) using the following formula:

$$(x_0, y_0, x_1, y_1) \rightarrow (x_c, y_c, H, W) \qquad \text{s.t.} \begin{cases} x_c = \frac{|x_0+x_1|}{2} \\ y_c = \frac{|y_0+y_1|}{2} \\ W = |x_0 - x_1| \\ H = |y_0 - y_1| \end{cases} \tag{1}$$

Connection determination

After generating the Flowmind components, the next task is to link the connectors to the keypoints on the shape. In both Visio and PPT, if a connector is generated solely based on coordinates, it will remain fixed in its original position even if the shape is moved. To establish a true relationship between the shape and the connector, it is essential to connect it with the pre-defined keypoints on the shape. Figure 12 illustrates the difference between the two methods of connector generation based on coordinates and connection.

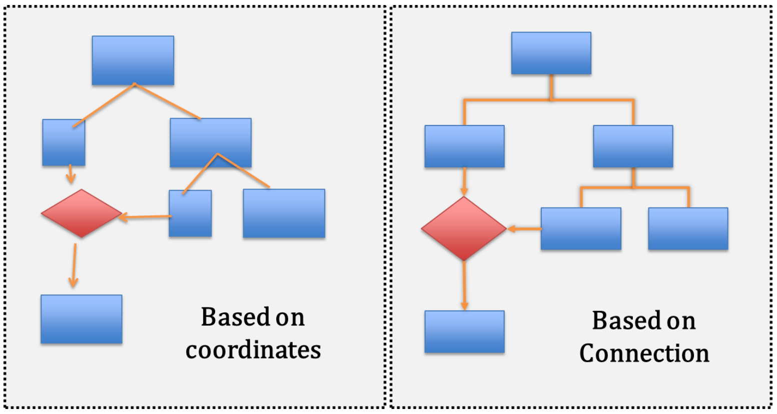


Figure 12. The difference of two generation methods based on coordinates and connection

Our algorithm aims to establish a connection between connectors and shapes by calculating the Euclidean distance between their keypoints. Euclidean distance serves as an intuitive and effective metric, accurately expressing the distance between keypoints

[10] https://pypi.org/project/pywin32/

of connectors and various geometric shapes. Its application facilitates the calculation of distances between connectors and candidate points on shapes, derived from the geometrical relationship between bounding boxes and standardized shapes. This approach, which includes calculating distances for polygons, identifying candidate points for non-polygons, and selecting the nearest shape for each keypoint, demonstrates robustness and adaptability in handling diverse geometric scenarios. The flexibility of Euclidean distance enhances the algorithm's ability to accurately model and understand geometric relationships, ensuring a robust and reliable approach for connector and shape connection in the recognition of complex hand-drawn sketches. The distance calculation between connectors and various types of geometric shapes can be derived using the geometrical relationship between the bounding box and the standardized shape, as shown in Fig. 13.

Assuming that the detected shapes in bounding box is set in an standardized orientation (with a horizontal base), the process first calculate the candidate points on each shape, referring to the connectable anchors on PPT and Visio shapes. Secondly for each connector keypoint, it identifies the nearest candidate point on all shapes. As for polygons, it computes the vertical distance from the keypoint to each edge (as depicted in Fig. 13a d_1, d_3). Note that, if the foot point of that vertical line lies on the extension of the edge, it chooses the shortest distance from the keypoint to terminal point of the edge as the shortest distance instead (as depicted in Fig. 13a d_2, d_4). For non-polygons, it identifies n candidate points on the shape (according to the connection rules of PPT and Visio) and specifies that the keypoint can only be connected to them. For instance, the candidate points of a circle are Up, Down, Left, Right, Top left, Bottom left, Top right, and Bottom right. Lastly, for each keypoint, the shape with the nearest candidate point is selected as the connected object (as depicted in Fig. 13b).

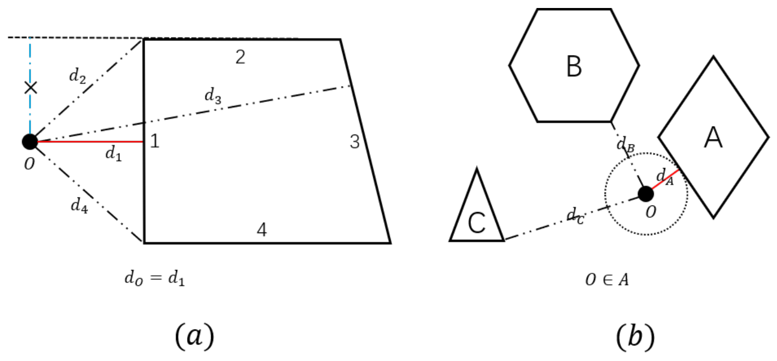


Figure 13. Calculation of distance between connectors and different types of geometry shapes

Text content

In this step, our approach aims to recognize specific content within the text boxes. The input consists of a set of coordinates for each textbox, T, and the image feature map, F. The output includes the recognized content set, C, and a confidence score, S. Instead of training an OCR model from scratch, we use an existing OCR model, which offers superior accuracy and speed. The OCR model is also a two-stage method that generates bounding boxes in regions of interest and then recognizes the specific content of each bounding box. However, due to the challenges mentioned in Section 3, its performance on hand-drawn flowminds is considerably limited. Therefore, we apply OCR in each text box identified previously, which greatly improves the accuracy of text recognition, as demonstrated in Section 6.

To address merging or splitting problems in text box recognition, we utilized the method proposed in Arrow-RCNN [4] to create a unified text box with a union bounding box that covers the corresponding text boxes. To determine which shape or connector a textbox corresponds to, we calculated the intersection over Union (IoU) between each text box and all bounding boxes of shapes. As mentioned in the shape recognition challenge, an IoU threshold of 80% was set. If there exists a shape that has the highest IoU rate over the threshold for a detected text box, its content is filled into that shape. Otherwise, it is considered as an independent text element. As connectors have high flexibility, we created the text box through the corresponding bounding box to fill in its content.

Automatic typesetting

The steps described earlier have established the inclusion and graph relations between shapes, connectors, and text boxes, primarily realizing the visualization. However, a precise copy of the rough sketch may not always reflect the user's intention. For instance, the rectangles in Fig. 14 are meant to be of the same size, but due to the visualization being based on the shape of the bounding box, there may be slight variations in the actual digitization. Additionally, the rectangles should be vertically aligned, but the digitized coordinates may not reflect this due to differences in the actual sketch.

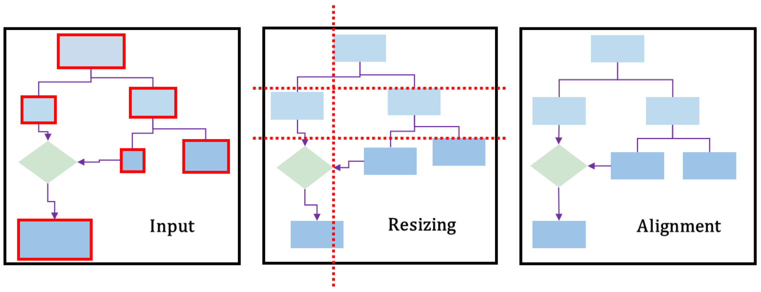


Figure 14. Automatic Typesetting: Intelligent Scaling of Shape Sizes, Followed by Automatic Horizontal and Vertical Alignment

The aforementioned deviation can pose problems, especially when creating a digital flowchart automatically. In a manually created flowchart, the consistency of a set of shapes can be ensured by copying and pasting. However, if the flowchart is already generated on the software, adjustments to individual shapes have to be made separately as deleting or replacing any shape can affect the established relations. Hence, an automatic typesetting algorithm that can assist in intelligent typesetting becomes particularly important.

To achieve this, we have implemented a two-stage clustering model that employs the Canopy and K-means algorithms. The number of clusters is determined through Canopy clustering, which is then followed by K-means to produce the final result. Moreover, we have utilized these clustering algorithms to adjust the size of shapes to account for variations in the input flowminds. Generally, the clustering algorithm is applied twice for automatic resizing and alignment. A summary of the clustering algorithm is depicted in Fig. 14.

To resize the editable graphics, we utilize a two-stage clustering model based on Canopy [cite] and K-means algorithm. Firstly, we use the length and width of the bounding box as clustering features. We set the thresholds of the Canopy algorithm and consider shapes with similar length and width as a cluster in coarse clustering. This provides us with a clustering reference value K. In the next stage, we perform fine-grained clustering using K-means algorithm, and calculate the average size of the bounding box for each cluster. This average size is used as the new size for the shape cluster. We perform this resizing process twice, to ensure automatic resizing and alignment of the shapes.

Moreover, Canopy clustering stands out by eliminating the need for a pre-specified k value, making it exceptionally practical. Despite potentially lower accuracy compared to other clustering methods, Canopy excels in speed, making it a valuable choice. Hence, Canopy clustering is strategically applied for preliminary coarse clustering, allowing the machine to autonomously determine the K value and approximate K initial centroids. Subsequently, this is followed by a more detailed fine clustering using K-means. The Canopy+K-means clustering strategy not only balances speed and accuracy but also proves effective in the context of automatic typesetting, emphasizing the machine's ability to specify the number of clusters in subsequent work.

To achieve alignment, the four coordinates of the bounding box of each shape cluster obtained in the first stage are used as clustering features. The horizontal and vertical coordinates are clustered separately using the same two-stage approach as for resizing. The average coordinates of each cluster are then determined as the final layout result. The specific threshold parameters for the Canopy algorithm will be described in more detail in Section 6.1.

6. Evaluation

To evaluate and analyze the performance of our methods, we trained and optimized the model on our *hdFlowmind* dataset. A detailed description about evaluation setup, experimental contents, results, and analysis will be provided in this section.

6.1. *Evaluation Setup*

This section will provide an in-depth explanation of the evaluation setup for our implementation, including the metrics and baseline used to assess the performance of our model. Researchers can access our code demo on GitHub[11].

Implementation

Our neural network is based on the framework of Detectron2, which utilizes Mask-RCNN and heat map keypoints detection with pytorch. For our experiments, we utilized the Keypoint-ResNet-50-FPN11 backbone, which is relatively fast and balances speed and accuracy effectively. We initialize the model weights using the pre-trained model from the COCO dataset, obtained from the Detectron2 model zoo. When it comes to hyperparameters, we mostly follow the default Detectron2 configuration for training. Specifically, the top-k anchor in train and test are 1500 and 1000 respectively in RPN, the base learning rate is 0.02, the smooth β in l_1 is 0.5. Please refer to Detectron2 configs for more hyperparameters and architecture settings. For gradient descent, we use Adam with 80k iterations and a batch size of 4, allowing the model to see 320k augmented images (80k batches of size 4). This process takes approximately 8 hours on a GeForce RTX 3090 with 16GB memory. During training, the basic learning rate is set to the default value of 0.00025 for Adam's adaptive change of Detection2.

For post-processing, we utilize Baidu's offline service paddleOCR[12], specifically the PP-OCRv3 version, which supports both Chinese and English languages. This service is based on the PaddlePaddle framework and prioritizes precision and speed balance. To achieve this, it employs model slimming and depth optimization techniques. Since it is deployed in an offline environment, users can choose whether or not to perform character recognition. For automatic typesetting, we use the T1 and T2 parameters of Canopy, as shown in Table 4. For Kmeans clustering, we use the default parameters of the sklearn module.

Table 4. Clustering parameters

Canopy	T1(inch)	T2(inch)
First Clustering	1	$\min \frac{length^2+width^2}{1.618}$
Second Clustering	0.8	$\min \frac{length}{1.618}, \min \frac{width}{1.618}$

Another crucial aspect of our method involves performing non-maximum suppression between different classes to address the issue of excessive IoU. To assess its impact within our specific domain, we conducted an ablation study, comparing two models: one

[11] https://github.com/cai-jianfeng/flowmind2digital.git
[12] https://www.paddlepaddle.org/

employing the default method from Detectron2, and the other utilizing an additional NMS between different classes after prediction to filter the results twice.

Furthermore, to evaluate the effect of adding single basic shapes to the training set, we conducted another ablation experiment that did not involve changing the parameters of other methods. This experiment involved comparing the changes in each metric before and after adding 485 images, as discussed in Section 4.

Metrics

Object detection: To assess the performance of object detection, we utilize the same metrics as in the relevant sketch recognition approach[4]. A bounding box is considered a true positive only if it is categorized correctly and overlaps fully with the ground-truth. We set the IoU threshold to 50%, following previous works [4][19]. We then use these true positives as the predicted object detection results and calculate the standard recall, precision, and F1 scores with an IoU threshold of 70% during the calculation process. Additionally, we calculate the diagram accuracy (DA) [4], which represents the proportion of images with completely correct object detection in all datasets, i.e., standard recall and precision are both 1 when the IoU threshold is 80%, and the number of precision boxes and ground-truth boxes is equal. Since some images have zero predicted objects, the calculation of precision can result in division by zero, causing F1 and precision to return N/A for a single image. These images are ignored when calculating the average value. Similarly, when zero annotated objects are selected, F1 and recall return N/A. However, this situation does not arise in the training data, so it does not impact the calculation of diagram metrics for training and validation sets. Finally, when precision and recall are both zero, F1 is defined as 0.

Connector Keypoints: For the evaluation of the performance for connector recognition, we refer to the relevant work [19]. The detection of keypoints by neural network may have some error in the circular domain. But in fact, we are concerned about the connectivity of keypoints with shapes, that is, whether a group $(x_{from}, y_{from}, x_{to}, y_{to})$ can correctly find the connection between the shapes. Therefore, we apply the same post-processing method to find the nearest shape for train and test, which is mentioned in Section 4.2 to obtain the ground-truth connection and predicted connection of a connector. Then the standard Recall, Precision and F1 scores of the connector category are calculated in the same way as the object detection. In this context, true positive is defined as the correct recognition and full overlap of the three connector categories, where the two connected shapes have the same precision and true label.

Character recognition: Section 5.2 discusses the comparison between OCR for the entire image and OCR for the identified text box, which is also an essential aspect. To assess the performance of both methods under various scenarios outlined in Challenges (Section 3), we utilize the character error rate CER [26], which is based on the editing distance [27], for each text box string. However, since the dataset contains mathematical formulas and symbols, there are no complete Chinese and English character labels available. Thus, we randomly select 50 representative images from the dataset and manually annotate them to evaluate the performance.

Baselines

In order to demonstrate the effectiveness of our approach, we conduct a comparative analysis with related studies. We find that BPMN with strict graphic definition, which is heavily focused on the professional domain, is not an ideal baseline. Instead, we choose Arrow-RCNN as it is similar to our work and has a hand-drawn sketch of almost all classes. Accordingly, we train and evaluate Arrow-RCNN models on the hdBPMN dataset to compare them with our *Flowmind2digital* model under various scenarios outlined in Section 3. For Arrow-RCNN, we adhere to its default image augmentation methods and training parameters.

In order to test our hypothesis about the dataset, we conducted an experiment where we used the same model and replaced various datasets to evaluate the effectiveness of our hdFlowmind. We divided the experiment into two groups: one group was trained using a pre-trained model on flowmind and fine-tuned on the Handwritten-diagram-dataset, while the other group was trained without a pre-trained model. Our objective was to demonstrate the importance of our dataset and show that pre-trained models can be beneficial for training even in different domains.

6.2. *Results*

This section presents the evaluation results of our proposed *Flowmind2digital* model and our *hdFlowmind* dataset. First of all, the overall results are displayed, followed by the detailed results for each part of our model. Next, we illustrate the ablation studies. Finally, the time-memory complexity analysis and post-processing software docking are presented.

Overall results and baselines

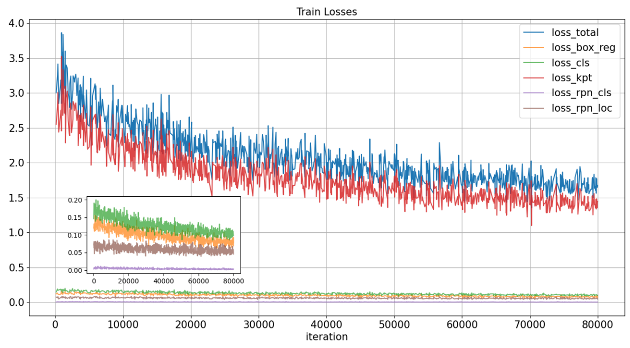


Figure 15. Losses During the Whole Training Process on hdFlowmind

Fig. 15 shows the individual loss terms and metrics throughout the 80k iterations on the *hdFlowmind* dataset. It can be clearly observed that the loss mainly stems from the localization of connector keypoints, and after 80k iterations of training, the initial intense fluctuations tend to become stable. The loss of the RPN network, bounding box regression and classification is below 0.2 and gradually tends to fit as the training progresses.

The overall results and metrics of evaluation are presented in Table ?? compare with Arrow-RCNN. For certain classes, the F1 score exceeds the range of Recall and Precision, due to one of them being N/A. In this case, the F1 score is also N/A and is not included in the average calculation, resulting in this outcome. Note that in the experiment, the calculation method of the four metrics is weighted average by the number of categories. As shown in Table ??, our approach has a better ability to capture complex scene features, with the total diagram accuracy that is 20% higher than that of Arrow-RCNN, and other metrics that are about 15% higher. Further comparison of recognition examples reveals that our method performs significantly better than Arrow-RCNN in scenarios with more noise, such as over-exposure and shadows. In terms of arrow recognition, Arrow-RCNN has poor performance in identifying multiple arrows or intersecting arrows. In terms of shape and text recognition, it struggles to distinguish styles with overlapping strokes. Overall, our method demonstrates stronger robustness and better performance in fine-grained recognition.

Next, we fine-tuned a pre-trained model using FC_A, FC_B, F_A and hdBPMN datasets on our hdFlowmind dataset. The performance of the fine-tuned models was compared with the models trained directly on these datasets without pre-training. The model architecture and parameters used in the training process were identical to those used in the training of *Flowmind2digital*. The iterations for both experimental groups was set to 5,000. From training process (Fig. 16), it can be inferred that using the *hdFlowmind*

Table 5. Overall approach results for the test set

	Shape			Connector			Text			Total			
Approach	Rec.	Prec.	F1	Rec.	Prec.	F1	Rec.	Prec.	F1	Rec.	Prec.	F1	Diagram accuracy
Arrow R-CNN [4]	0.7875	0.8250	0.8057	0.7342	0.6964	0.7147	0.7789	0.7472	0.7627	0.764	0.718	0.754	0.2%
Flowmind2digital	0.9698	0.9544	0.9760	0.8085	0.7577	0.8136	0.8472	0.8247	0.8407	0.885	0.865	0.873	22.5%

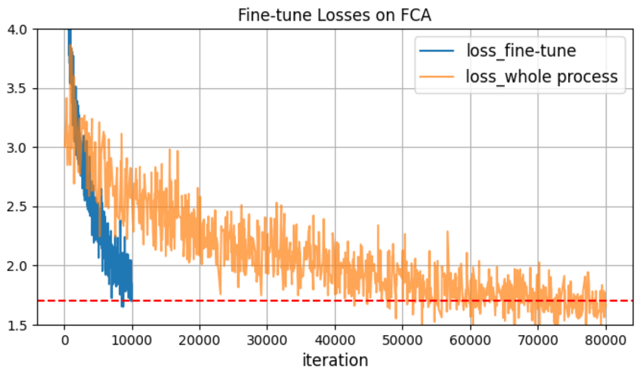


Figure 16. Comparison of Training Curves with and without Fine-tuning

pre-trained model for fine-tuning on the FC_A dataset greatly accelerates gradient descent and loss convergence compared to not using a pre-trained model. Subsequently, we present a comparison experiment (Table 6) for fine-tuning on the aforementioned dataset. After conducting multiple experiments and taking the average results, it was found that using the pre-trained model provides a certain degree of improvement in F1 score and diagram accuracy, especially for the FCA and FCB datasets, which are both process diagrams, with an improvement of up to 6%. For the FA dataset in the finite automata domain, due to the small number of categories, it quickly fitted with only 5,000 iterations, and the pre-trained model improved the metrics by approximately 1-2%. However, for the BPMN scenario with strict symbol standards, the 5,000 iterations were clearly insufficient for the model converge, and the hdBPMN2021 dataset's lack of text categories resulted in only a 1-2% improvement in recognition accuracy with the pre-trained model. Overall, our pre-trained model exhibits good generalization in the sketch domain, fully demonstrating the richness and diversity of the scenarios in our dataset.

Table 6. Comparison of metric results for fine-tuning experiment on different test set of our model

Dataset	**Pre-train on *hdFlowmind***		**No fine-tuning**	
	F1	DA	F1	DA
FA	0.9492	0.2143	0.9351	0.1305
FCA	0.8352	0.0585	0.8064	0.0350
FCB	0.9066	0.1326	0.8425	0.0408
hdBPMN	0.4347	-	0.4251	-
*DA=Diagram Accuracy				

Object detection

Table 7 reports a detailed performance of our object detection classes in a manner that describes the weighted average metrics of each shape. The results of our experiments indicate that our model is capable of accurately recognizing the majority of the primary shapes with high accuracy, achieving a recall rate, precision rate, and F1 score around 95%. For certain categories, due to their low occurrence in the test set, the aforementioned issues

Table 7. Object detection results per class obtained for the test set of our model

Class	Rec.	Prec.	F1	DA	Count
Circle	0.979	1.000	0.986	0.935	209
Diamond	0.985	0.978	0.980	0.940	96
Long oval	0.992	0.980	0.981	0.933	97
Hexagon	0.994	1.000	0.996	0.949	111
Parallelogram	0.959	0.991	0.969	0.893	122
Rectangle	0.986	0.961	0.968	0.720	465
Trapezoid	0.977	0.970	0.971	0.940	71
Triangle	0.981	0.976	0.974	0.902	127
WA	0.9824	0.97807	0.976	0.8525	1298
*DA = Diagram accuracy, WA = Weighted average					

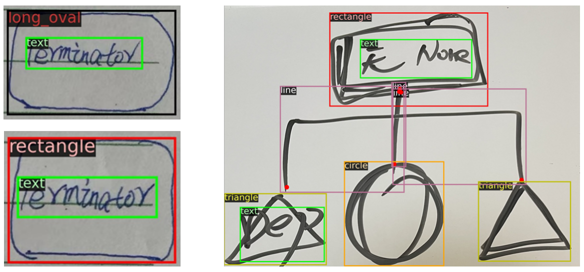


Figure 17. Left: Shapes that are extremely easy for humans to confuse (long ovals and rounded rectangles). Right: The recognition result effectively solves the challenges mentioned earlier.

related to F1 calculation surpass recall and precision rates, and thus, further evaluation is needed.

Post-hoc analysis of the results reveals that our target detector performs excellent in different backgrounds and handwriting scenarios, and it can handle the difficulties mentioned in Section 3, such as overlapping circles, crossing, and text overprinting. However, the most challenging task is to correctly distinguish certain categories, especially the confusion between rounded rectangles and long ovals, as depicted in the Fig. 17. This is not surprising, given the subjectivity of the drawings, and recognizing these differences between hand-drawn models is also a difficult task for humans, especially in terms of the variations in curvature.

Connector recognition

Table 8. Connector and keypoints detection results per class obtained for the test set of our model

Class	Rec.	Prec.	F1	DA	Count
Arrow	0.848	0.836	0.836	0.559	611
Line	0.855	0.845	0.845	0.714	157
Double arrow	0.750	0.744	0.743	0.525	264
WA	0.8240	0.8138	0.8136	0.5739	1032
*DA = Diagram accuracy, WA = Weighted average					

Table 8 further demonstrates the superiority of our model and its versatility in post-processing determination of connection as a whole. According to the first three metrics in the experimental results, arrows and lines are more easily recognized than double arrows, but the diagram accuracy is skewed due to the low number of line samples in the test set. The detection performance of double-arrows is not as good as the other two, which can be attributed to the triangular lines at the arrowhead that often overlap with shapes or are perceived as part of other shapes. Furthermore, the double arrow possesses features of both arrow and line, which increases the difficulty of recognition. This provides insight into future improvement efforts.

Upon further analysis, our recognizer is able to handle various challenges such as many-to-many connectors that cross the entire diagram with relatively good performance. However, for connections that are very close to each other, our model demonstrates limited ability to handle and may treat additional strokes as part of the error or as combinations of other shapes, as shown in Fig. 18. A reasonable explanation for this phenomenon lies in the intricacies of proximity-based spatial relationships. When connectors are very close, the model might face difficulties in precisely separating them, as the visual features become more challenging to differentiate. Additionally, the proximity of strokes may introduce ambiguity, leading to potential errors in the recognition process.

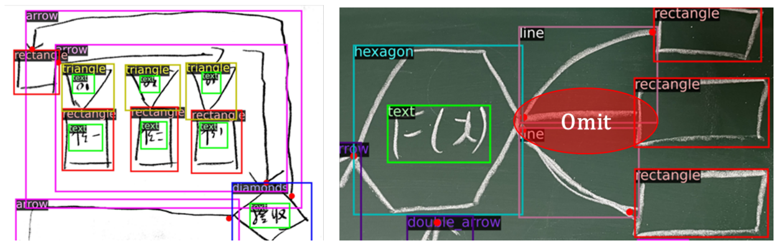


Figure 18. Left: Successfully recognized large-span arrows and various fine-grained graphics. Right: Missed the short connecting line selected by the red oval.

In conclusion, our method consistently achieves metrics approaching 80% on connectors, highlighting its efficacy in handling intricate visual relationships, while the recognition performance for connectors may be comparatively lower than that of shapes owing to their inherent flexibility and diversity. This underscores the distinct advantages of our approach, particularly in addressing the complexities of scenarios outlined in Section 3, such as crossings.

Textbox recognition

Table 9. Comparison of using textbox selection and directly using OCR

Class	**Rec.**	**Prec.**	**F1**	**Diagram accuracy**
Text box	0.8509	0.8354	0.8407	0.2913
Metric	**Specified region**		**Whole region**	
CER	8.5%		35.7%	

As shown in Table 9, the results of employing the Baidu PaddleOCR service for handwriting recognition are poor. As demonstrated in Section 3, the main reason of low accuracy is the inability to distinguish between handwritten text and graphical intersections. However, this is not the case for our method, as the first stage of regions of interest identification directly frames the scope. At this point, OCR is used to decode the text in the specified region, and CER has seen a significant improvement, performing well on isolated text boxes. This means that if the OCR prediction is accurate and can reach the level of humans, the detected text block (region of interest) will also be accurate.

It should be noted that our method only achieves a recognition accuracy of 29% for text boxes in images, particularly for the recognition of mathematical symbols, brackets, operators, etc. This warrants further investigation.

In conclusion, the majority of errors directly or indirectly incurred during the handwritten text block recognition process are due to OCR service errors, and fine-tuning of the ROI plays a significant role. However, this does not indicate that existing OCR services have major flaws, because high accuracy can still be achieved for pure text recognition in small areas.

Ablation study

The results of our ablation study in Table 10 show the benefits of adding images of single basic shapes to the training set, improving the Recall (from 0.792 to 0.889), Precision (from 0.784 to 0.872) and F1-score (from 0.786 to 0.879) to increase by 10%. In particular, it greatly improves the diagram accuracy over 20%. From the training process, these images greatly alleviates the over-fitting situation during model training, especially for connectors. It is, therefore, not necessary to excessively pursue sophisticated images when collecting datasets, and single target recognition is also promotional for machine learning.

Table 10. Ablation study conducted on the validation set

Method	Rec.	Prec.	F1	Diagram accuracy
NMS	0.792	0.784	0.786	0.066
Add basic images	0.894	0.864	0.877	0.275
NMS+Add basic images	0.889	0.872	0.879	0.283

The results of applying NMS between different classes is not as effective as the former. However, it is obvious from Table 10 whether NMS between different classes improves Precision-score and reduces Recall-score around 0.01. Further, this also improves the diagram accuracy. This experimental result is consistent with our intuition. NMS reduces the number of redundant bounding boxes, and also removes true positive bounding boxes, leading to the decline of recall rate, and vice versa, resulting in the improvement of precision. In the field of sketch, because of the simple shape and the arbitrariness of the user, it is common for the same basic strokes to have two categories of high accuracy at the same time. In the case of multiple connectors, additional detected lines are often generated due to crossing and masking. Therefore, in the domain of the flowmind, this assumption works effectively. Fig. 19 shows an example of how this method works on sketches.

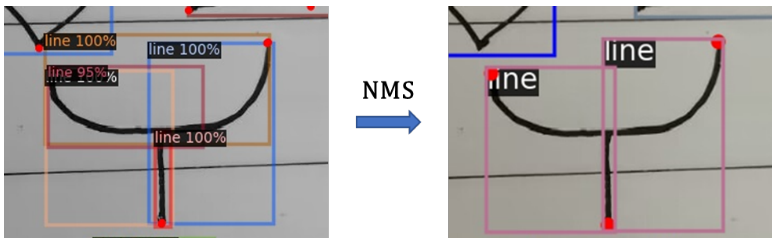


Figure 19. Non-maximum suppression between different categories effectively solves the problem of many-to-many connectors being recognized multiple times and overlapping.

Running time and memory

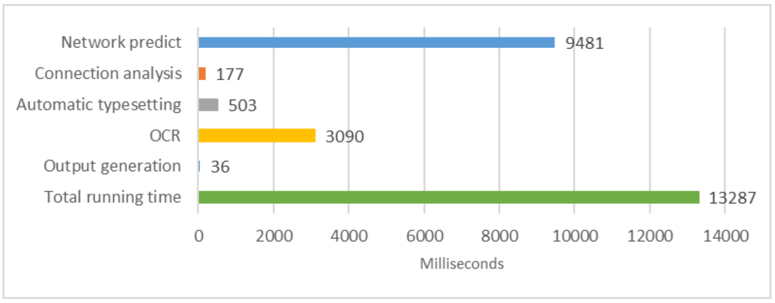


Figure 20. Median runtime measures obtained for the validation set

Fig. 20 illustrates the runtime of the components of *Flowmind2digital*. Given the inconvenience of installing the Conda environment for actual users, we tested using a CPU (Interl(R) Core(TM) i5-8300H CPU @ 2.30GHz). Given the image to be processed (3000*4000 pixels), it first goes through the network prediction module (9481ms), followed by the relation analysis and graph construction (177ms), automatic typesetting (503ms), and the optional OCR module (3090ms), and finally the integrated output (36ms). The whole process took 13287ms, with most of the time spent on the two neural modules. The observed processing time of approximately 10 seconds represents a favorable trade-off between accuracy and speed on a standard CPU. This duration is considered a well-balanced solution for practical applications, aligning with the need to efficiently process and interpret visual information without compromising on detection precision. If we use GPU (e.g. GeForce RTX 3090) in inference, the whole process can be completed within one second.

Fig. 21 shows the memory usage of *Flowmind2digital* over time. In monitoring memory usage, it is noteworthy that the initialization time of Python-related modules extends the overall processing time beyond the 10-second mark. The peak memory is around 3500MB at 15.3s, which occurs during the final integration of the module outputs. Most of the memory consumption still occurs in the neural network module, in line with the time consumption.

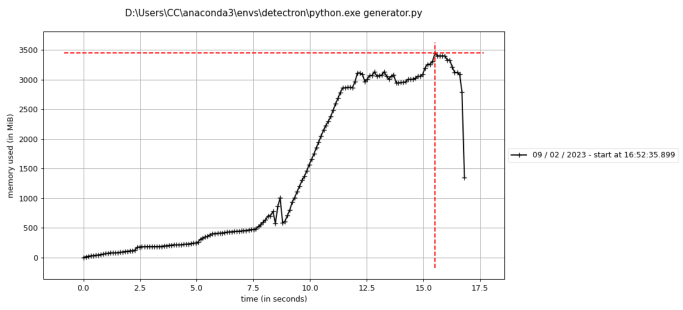


Figure 21. Memory Analysis of the Whole Process

Software Results Display

Finally, we display the final result of the integration of the user-inputted raw sketch image with the PPT/Visio software. Note that, based on the flexibility and standardization of both software, our result can only serve as a preliminary reference for further refinement by the designer. In PPT, the shape color is based on the principle of category consistency, while the texture and outline filling follow the default template. In Visio, we initially design the interface in blue as the main color scheme. The font and line width adapt to the size of the input graphic and the bounding boxes. The output visualization is shown in Fig. 22.

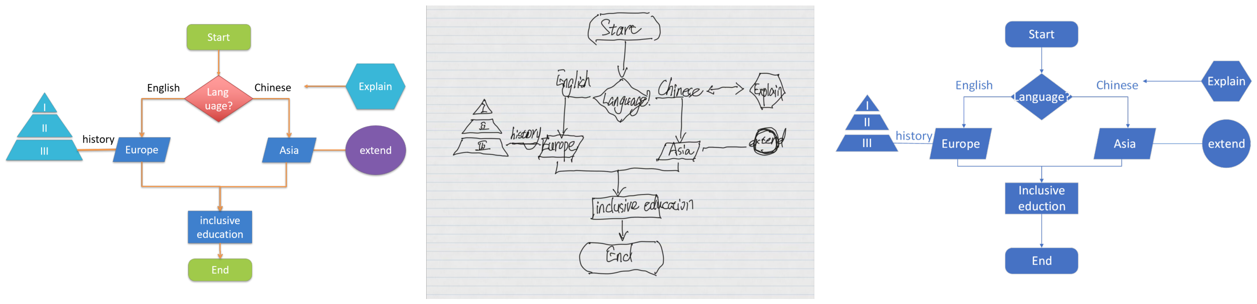


Figure 22. Visualization of the Final Software Results

7. Discussion

In this section, we discuss the implications, limitations, and provide reference insights for future research work.

Implications. Our work extends from the specific domain of hand-drawn diagrams, such as BPMN, UML, and flowcharts, and generalizes further by considering the characteristics of natural hand-drawn diagrams. We combine the multi-connection features of mind maps with the geometric shapes that are commonly used and creatively varied in composition to better reflect the natural hand-drawn features.

Moreover, our approach establishes a more novel, convenient, and effective comprehensive processing mode. It enables brainstorming and creation on a whiteboard or even a blank sheet of paper, followed by direct digital acquisition, collection, and storage of composition information. Connection analysis and automatic layout are then performed, and finally, a document is generated using OCR services. The entire process is fully automated and intelligent. It is worth noting that because the OCR service is independent, the text recognition performance can be further improved as existing natural language processing models evolve in the future.

In addition, we have created the *hdFlowmind* dataset, which is the first object and keypoint detection dataset covering both mind maps and flowcharts. It contains thousands of images and tens of thousands of annotations, and has a wide range of applications. It can effectively address the problems mentioned in the challenge. Furthermore, we found that adding simple basic shapes to the dataset can effectively solve the overfitting problem during the training process for hand-drawn sketches (without stroke sequence information), which contain far less information than RGB images.

Limitations. Objectively speaking, our model also has a series of limitations. Firstly, our post-processing relies on specific software interfaces, e.g. if Microsoft PPT/Visio is no longer used as the most common presentation software, the post-processing work will need to be further modified for integration. To expand the model's application scope and integration into standardized BPMN scenarios, professional visualization software needs to be integrated. Even when the software does not provide ready-to-use documentation, library functions need to be written at the operating system level.

Secondly, although we believe that our dataset scenarios have high external validity and cover a variety of application scenarios, the images in the dataset are mainly collected from various scenarios in university life, and the overall image features and quality differ significantly. Therefore, in practical use, people may still encounter more complex situations, such as multiple people using pens of various colors to draw on the same whiteboard in a messy way.

Finally, limited by the diversity of connector paths, our model can only recognize the connection relationships and cannot distinguish between the forms of a path, such as straight or curved arrows. Moreover, the electronic version of the connector also depends on the template of the relevant visualization software and cannot truly fit the path of the hand-drawn arrow. This is also a major bottleneck in all sketch recognition works and requires further research in the future. At the same time, different expressions of the same geometric figure is also be a big challenge, which can enrich our graphic material library, such as isosceles triangle, right triangle, etc.

8. Conclusion

In this paper, we focus on the recognition problem of hand-drawn sketches. We combine the characteristics of flowcharts and mind maps, and propose our *Flowmind2digital* method on the basis of existing solutions. By introducing OCR and integrating with Microsoft Power Point/Visio visualization software, it is the first comprehensive, fully automated sketch recognition method. To enrich application scenarios and better fit the characteristics of hand-drawn sketches by natural persons, we created the "*hdFlowmind*" dataset, which consists of 1776 images and tens of thousands of annotations, solving the recognition difficulties encountered in actual use due to messy sketches. In the experimental evaluation process, we not only demonstrated the effectiveness of the *hdFlowmind* dataset, but also showed that *Flowmind2digital* is very accurate, versatile in use scenarios, capable of handling fine-grained recognition problems in complex sketches, and consistently outperforms existing algorithms.

Next, we have identified several directions for future work. First and foremost, increasing recognition accuracy and speed are undoubtedly the most importance. On the one hand, we can continue to enrich the breadth of the dataset, covering a wider variety of scenarios and increasing the number of shape categories. On the other hand, for connector recognition, research can be conducted based on its path characteristics. In text recognition, further collaboration with the NLP field will be pursued to handle more detailed information such as rotation angles of handwritten characters. Second, exploring the combination of the flowmind pre-trained model with existing research in other symbol recognition fields (such as BPMN) is also an interesting topic. The merging of research in various fields of offline sketch recognition will help deepen our understanding of the universality of human sketches (e.g., connection relationships). Finally, increasing the practicality of *Flowmind2digital* in practice is also necessary. In the future, it can be inherited by more software in various fields, providing a variety of intelligent visualization solutions.

References

1. Gosala, B.; Chowdhuri, S.R.; Singh, J.; Gupta, M.; Mishra, A. Automatic classification of UML class diagrams using deep learning technique: convolutional neural network. *Applied Sciences* **2021**, *11*, 4267.
2. Schäfer, B.; van der Aa, H.; Leopold, H.; Stuckenschmidt, H. Sketch2BPMN: Automatic recognition of hand-drawn BPMN models. In Proceedings of the Advanced Information Systems Engineering: 33rd International Conference, CAiSE 2021, Melbourne, VIC, Australia, June 28–July 2, 2021, Proceedings. Springer, 2021, pp. 344–360.
3. Schäfer, B.; van der Aa, H.; Leopold, H.; Stuckenschmidt, H. Sketch2BPMN: Automatic recognition of hand-drawn BPMN models. In Proceedings of the Advanced Information Systems Engineering: 33rd International Conference, CAiSE 2021, Melbourne, VIC, Australia, June 28–July 2, 2021, Proceedings. Springer, 2021, pp. 344–360.
4. Schäfer, B.; Keuper, M.; Stuckenschmidt, H. Arrow R-CNN for handwritten diagram recognition. *International Journal on Document Analysis and Recognition (IJDAR)* **2021**, *24*, 3–17.
5. Awal, A.M.; Feng, G.; Mouchere, H.; Viard-Gaudin, C. First experiments on a new online handwritten flowchart database. In Proceedings of the Document Recognition and Retrieval XVIII. SPIE, 2011, Vol. 7874, pp. 81–90.
6. Bresler, M.; Průša, D.; Hlaváč, V. Online recognition of sketched arrow-connected diagrams. *International Journal on Document Analysis and Recognition (IJDAR)* **2016**, *19*, 253–267.
7. Gervais, P.; Deselaers, T.; Aksan, E.; Hilliges, O. The DIDI dataset: digital ink diagram data. *arXiv preprint arXiv:2002.09303* **2020**.
8. Yu, B.; Cai, S. A domain-independent system for sketch recognition. In Proceedings of the Proceedings of the 1st international conference on Computer graphics and interactive techniques in Australasia and South East Asia, 2003, pp. 141–146.
9. Chen, Q.; Grundy, J.; Hosking, J. SUMLOW: early design-stage sketching of UML diagrams on an E-whiteboard. *Software: Practice and Experience* **2008**, *38*, 961–994.
10. Brieler, F.; Minas, M. A model-based recognition engine for sketched diagrams. *Journal of Visual Languages & Computing* **2010**, *21*, 81–97.
11. Paulson, B.; Hammond, T. Paleosketch: accurate primitive sketch recognition and beautification. In Proceedings of the Proceedings of the 13th international conference on Intelligent user interfaces, 2008, pp. 1–10.
12. Julca-Aguilar, F.; Mouchère, H.; Viard-Gaudin, C.; Hirata, N.S. A general framework for the recognition of online handwritten graphics. *International Journal on Document Analysis and Recognition (IJDAR)* **2020**, *23*, 143–160.
13. Wu, J.; Wang, C.; Zhang, L.; Rui, Y. Offline Sketch Parsing via Shapeness Estimation. In Proceedings of the IJCAI. Citeseer, 2015, Vol. 15, pp. 1200–1206.
14. Costagliola, G.; De Rosa, M.; Fuccella, V. Local context-based recognition of sketched diagrams. *Journal of Visual Languages & Computing* **2014**, *25*, 955–962.

15. Bresler, M.; Van Phan, T.; Prusa, D.; Nakagawa, M.; Hlavác, V. Recognition system for on-line sketched diagrams. In Proceedings of the 2014 14th International Conference on Frontiers in Handwriting Recognition. IEEE, 2014, pp. 563–568.
16. Bresler, M.; Průša, D.; Hlaváč, V. Recognizing off-line flowcharts by reconstructing strokes and using on-line recognition techniques. In Proceedings of the 2016 15th International Conference on Frontiers in Handwriting Recognition (ICFHR). IEEE, 2016, pp. 48–53.
17. Ren, S.; He, K.; Girshick, R.; Sun, J. Faster r-cnn: Towards real-time object detection with region proposal networks. *Advances in neural information processing systems* **2015**, *28*.
18. Julca-Aguilar, F.D.; Hirata, N.S. Symbol detection in online handwritten graphics using faster R-CNN. In Proceedings of the 2018 13th IAPR international workshop on document analysis systems (DAS). IEEE, 2018, pp. 151–156.
19. Schäfer, B.; Van der Aa, H.; Leopold, H.; Stuckenschmidt, H. Sketch2Process: End-to-end BPMN Sketch Recognition Based on Neural Networks. *IEEE Transactions on Software Engineering* **2022**.
20. Lin, T.Y.; Dollár, P.; Girshick, R.; He, K.; Hariharan, B.; Belongie, S. Feature pyramid networks for object detection. In Proceedings of the Proceedings of the IEEE conference on computer vision and pattern recognition, 2017, pp. 2117–2125.
21. He, K.; Gkioxari, G.; Dollár, P.; Girshick, R. Mask r-cnn. In Proceedings of the Proceedings of the IEEE international conference on computer vision, 2017, pp. 2961–2969.
22. Amjoud, A.B.; Amrouch, M. Object Detection Using Deep Learning, CNNs and Vision Transformers: A Review. *IEEE Access* **2023**.
23. Zou, Z.; Chen, K.; Shi, Z.; Guo, Y.; Ye, J. Object detection in 20 years: A survey. *Proceedings of the IEEE* **2023**.
24. Redmon, J.; Divvala, S.; Girshick, R.; Farhadi, A. You only look once: Unified, real-time object detection. In Proceedings of the Proceedings of the IEEE conference on computer vision and pattern recognition, 2016, pp. 779–788.
25. Carion, N.; Massa, F.; Synnaeve, G.; Usunier, N.; Kirillov, A.; Zagoruyko, S. End-to-end object detection with transformers. In Proceedings of the European conference on computer vision. Springer, 2020, pp. 213–229.
26. Sanchez, J.A.; Romero, V.; Toselli, A.H.; Villegas, M.; Vidal, E. ICDAR2017 competition on handwritten text recognition on the READ dataset. In Proceedings of the 2017 14th IAPR international conference on document analysis and recognition (ICDAR). IEEE, 2017, Vol. 1, pp. 1383–1388.
27. Ukkonen, E. Finding approximate patterns in strings. *Journal of algorithms* **1985**, *6*, 132–137.
28. Hammond, T.; Davis, R. Tahuti: A geometrical sketch recognition system for uml class diagrams. In *ACM SIGGRAPH 2006 Courses*; 2006; pp. 25–es.
29. Fang, J.; Feng, Z.; Cai, B. DrawnNet: offline hand-drawn diagram recognition based on keypoint prediction of aggregating geometric characteristics. *Entropy* **2022**, *24*, 425.
30. Montellano, C.D.B.; Garcia, C.O.F.C.; Leija, R.O.C. Recognition of Handwritten Flowcharts using Convolutional Neural Networks. *International Journal of Computer Applications* **2022**, *184*, 37–41. https://doi.org/10.5120/ijca2022921969.
31. Schäfer, B.; Stuckenschmidt, H. DiagramNet: hand-drawn diagram recognition using visual arrow-relation detection. In Proceedings of the Document Analysis and Recognition–ICDAR 2021: 16th International Conference, Lausanne, Switzerland, September 5–10, 2021, Proceedings, Part I 16. Springer, 2021, pp. 614–630.
32. Pittke, F.; Leopold, H.; Mendling, J. Automatic detection and resolution of lexical ambiguity in process models. *IEEE Transactions on Software Engineering* **2015**, *41*, 526–544.
33. Chakraborty, S.; Sarker, S.; Sarker, S. An exploration into the process of requirements elicitation: A grounded approach. *Journal of the association for information systems* **2010**, *11*, 1.
34. Yang, J.; Lu, J.; Lee, S.; Batra, D.; Parikh, D. Graph r-cnn for scene graph generation. In Proceedings of the Proceedings of the European conference on computer vision (ECCV), 2018, pp. 670–685.
35. Davis, B.; Morse, B.; Cohen, S.; Price, B.; Tensmeyer, C. Deep visual template-free form parsing. In Proceedings of the 2019 International Conference on Document Analysis and Recognition (ICDAR). IEEE, 2019, pp. 134–141.