# DELOC: Document Element Localizer

**Hammad Ayyubi**[1*], **Puneet Mathur**[2], **Md Mehrab Tanjim**[2], **Vlad I Morariu**[2]
[1]Columbia University, [2]Adobe Research,
**Correspondence:** hayyubi@cs.columbia.edu

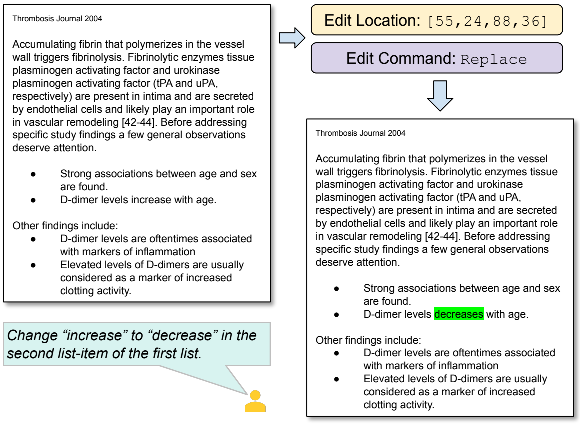
## Abstract

Editing documents and PDFs using natural language instructions is desirable for many reasons – ease of use, increasing accessibility to non-technical users, and for creativity. To do this automatically, a system needs to first understand the user's intent and convert this to an executable plan or command, and then the system needs to identify or localize the elements that the user desires to edit. While there exist methods that can accomplish these tasks, a major bottleneck in these systems is the inability to ground the spatial edit location effectively. We address this gap through our proposed system, DELOC (Document Element LOCalizer). DELOC adapts the grounding capabilities of existing Multimodal Large Language Model (MLLM) from natural images to PDFs. This adaptation involves two novel contributions: 1) synthetically generating PDF-grounding instruction tuning data from partially annotated datasets; and 2) synthetic data cleaning via Code-NLI, an NLI-inspired process to clean data using generated Python code. The effectiveness of DELOC is apparent in the >2x zero-shot improvement it achieves over the next best MLLM, GPT-4o.

## 1 Introduction

Editing documents and PDFs via natural language (Mathur et al., 2023; Suri et al., 2024) is an innovative and user-friendly advancement. This technology makes it easier for non-technical people to edit PDFs, speeds up the editing process, and facilitates document editing on a mobile device.

Typically, a two-stage process has been followed to edit PDFs from natural language requests. In the first stage, a model is required to predict the edit location (green highlight in Figure 1) spatially in the PDF. The edit location is a PDF element (paragraph, line, list, table, etc.). The second stage involves predicting the edit command (*replace* in Figure 1). The predicted bounding box and edit command can then be simply combined to effect the desired change. Existing PDF editing systems achieve high accuracy (>85%) in predicting edit commands (Mathur et al., 2023) but struggle with bounding box precision (<50%) (Suri et al., 2024). As such, we focus on improving the spatial PDF element localization for a given edit request.



Thrombosis Journal 2004

Accumulating fibrin that polymerizes in the vessel wall triggers fibrinolysis. Fibrinolytic enzymes tissue plasminogen activating factor and urokinase plasminogen activating factor (tPA and uPA, respectively) are present in intima and are secreted by endothelial cells and likely play an important role in vascular remodeling [42-44]. Before addressing specific study findings a few general observations deserve attention.

- Strong associations between age and sex are found.
- D-dimer levels increase with age.

Other findings include:
- D-dimer levels are oftentimes associated with markers of inflammation
- Elevated levels of D-dimers are usually considered as a marker of increased clotting activity.

*Change "increase" to "decrease" in the second list-item of the first list.*

Edit Location: `[55,24,88,36]`

Edit Command: `Replace`

Thrombosis Journal 2004

Accumulating fibrin that polymerizes in the vessel wall triggers fibrinolysis. Fibrinolytic enzymes tissue plasminogen activating factor and urokinase plasminogen activating factor (tPA and uPA, respectively) are present in intima and are secreted by endothelial cells and likely play an important role in vascular remodeling [42-44]. Before addressing specific study findings a few general observations deserve attention.

- Strong associations between age and sex are found.
- D-dimer levels decreases with age.

Other findings include:
- D-dimer levels are oftentimes associated with markers of inflammation
- Elevated levels of D-dimers are usually considered as a marker of increased clotting activity.

Figure 1: Illustration of a typical PDF edit process. Models predict Edit Location and Edit Command, which are combined to effect the required edit. This work focuses on improving the bottleneck step – Edit Location. To this end, the proposed system, DELOC, generates large-scale (3M) synthetic data to instruction tune a Multimodal LLM.

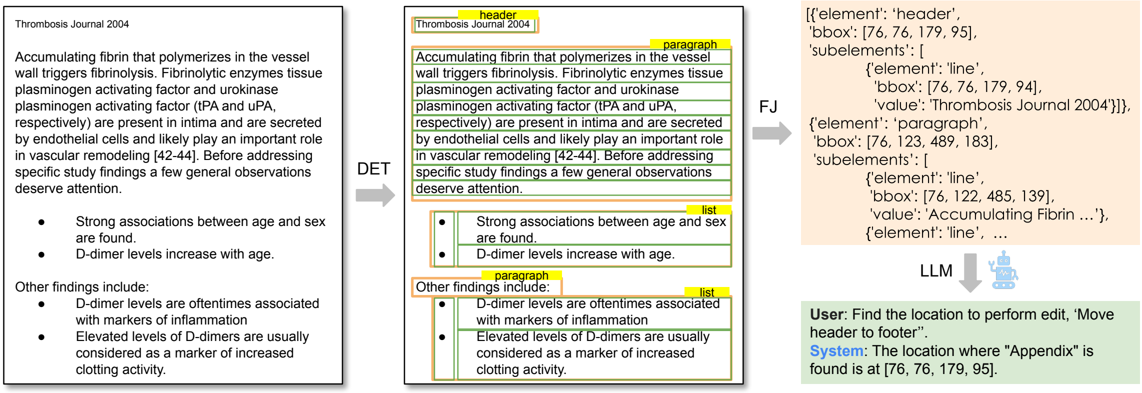
State-of-the-art systems ground user queries in images (Wang et al., 2023; You et al., 2023), mobile UIs (You et al., 2024), and web UIs (Hong et al., 2023), but they do not transfer directly to PDFs. PDFs are more verbose, and edit requests follow a hierarchical structure requiring domain knowledge. For instance, a request like "first list item of the third list in Section A.1" demands an understanding of the composition: section → list → list item.

To address these challenges, we propose DELOC (Document Element LOCalizer), which adapts the strong grounding capabilities of Mul-

---

[*] Work done during an internship at Adobe Research, Document Intelligence Lab (DIL)

*Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 31138–31147
November 4-9, 2025 ©2025 Association for Computational Linguistics



Thrombosis Journal 2004

Accumulating fibrin that polymerizes in the vessel wall triggers fibrinolysis. Fibrinolytic enzymes tissue plasminogen activating factor and urokinase plasminogen activating factor (tPA and uPA, respectively) are present in intima and are secreted by endothelial cells and likely play an important role in vascular remodeling [42-44]. Before addressing specific study findings a few general observations deserve attention.

- Strong associations between age and sex are found.
- D-dimer levels increase with age.

Other findings include:

- D-dimer levels are oftentimes associated with markers of inflammation
- Elevated levels of D-dimers are usually considered as a marker of increased clotting activity.

```
[{'element': ‘header’,
 'bbox': [76, 76, 179, 95],
 'subelements’: [
          {'element': 'line’,
            'bbox': [76, 76, 179, 94],
            'value': 'Thrombosis Journal 2004'}]},
 {'element’: ‘paragraph’,
  'bbox': [76, 123, 489, 183],
  'subelements’: [
          {'element': 'line’,
           'bbox': [76, 122, 485, 139],
           'value': 'Accumulating Fibrin …’},
          {'element': 'line’,  …
```

**User**: Find the location to perform edit, ‘Move header to footer”.
**System**: The location where "Appendix" is found is at [76, 76, 179, 95].

Figure 2: Overview of DELOC. Document elements, their bounding box, and their hierarchical relations are first detected (DET) and formatted into a JSON structure (FJ). This JSON is fed to an LLM to generate synthetic data.

timodal Large Language Models (MLLMs) from natural images to PDFs. This adaptation involves instruction-tuning an MLLM specifically for PDFs. Our key contribution is the synthetic generation of high-quality instruction-tuning data, followed by an automated cleaning process called Code-NLI.

We first represent PDF elements (paragraphs, lines, lists, tables, titles, sections) in a spatially aware hierarchical JSON format (Figure 2), capturing each element's bounding box and hierarchical relationships (e.g., paragraph → lines, list → list items). This information is sourced from partially annotated PDF Document Analysis datasets (e.g., PubLayNet (Zhong et al., 2019)), using existing annotations and heuristically generating missing ones. Next, we input this structured PDF representation into a Large Language Model (LLM) to generate synthetic user requests for PDF editing, along with system responses specifying edit locations via bounding boxes. The hierarchical structure enables the LLM to produce compositional requests that resemble real user queries, while the bounding box data ensures precise edit location predictions.

We further clean noisy generated samples using a proposed process called Code-NLI (Code-based Natural Language Inference). Code-NLI (Figure 3) treats the generated user edit request and system response as the hypothesis, and the PDF as the premise. To verify the hypothesis, it leverages an LLM to generate Python code, which is then executed to filter out noisy samples.

To demonstrate the effectiveness of DELOC, we evaluate it on DocEdit (Mathur et al., 2023). Our results demonstrate that DELOC outperforms all existing zero-shot models on PDFs, including proprietary MLLM GPT-4o by >2x. Our ablations demonstrate DELOC's performance improves with data scale and data cleaning via Code-NLI is crucial to its performance.

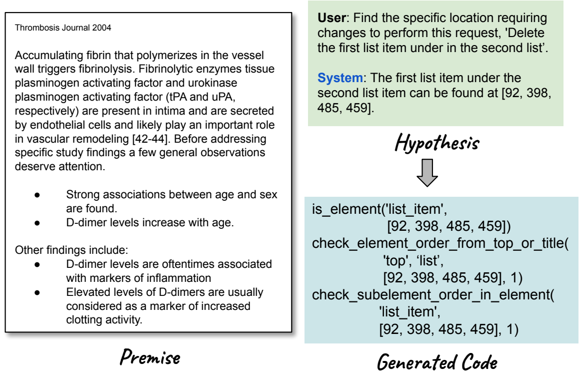
## 2 Related Work

There has been a surge of interest in natural language based PDF edit localization. DocEdit (Mathur et al., 2023) trains a multimodal transformer that directly regresses the edit bounding box given the document as input. DocEdit-v2 (Suri et al., 2024) trains an encoder-decoder architecture that outputs a segmentation mask for the edit location. In training a model from scratch, these methods fail to utilize the excellent grounding capabilities of recent MLLMs.

A number of MLLMs (Chen et al., 2023; Zhang et al., 2023; Yuan et al., 2023; Lv et al., 2023; Lai et al., 2024; Ma et al., 2024) have been proposed that have shown impressive grounding abilities on natural images. Works like Ferret-UI (You et al., 2024), Cog-Agent (Hong et al., 2023), and MultiUI (Liu et al., 2024) have successfully adapted these MLLMs for mobile screens and web pages. We take inspiration from these works to adapt MLLMs for PDFs using large-scale synthetic data. In comparison, Agent-DocEdit (Wu et al., 2024) finetunes a grounding model, GLIP (Li* et al., 2022), only on a small training set.

## 3 Method

Our method adapts the excellent query understanding and grounding capabilities of existing Multimodal Large Language Models (MLLMs) for grounding edit queries in PDFs. To this end, we: 1) generate diverse, compositional, and clean instruction-tuning data; and, 2) instruction-tune a MLLM on the generated data. We describe the synthetic data generation process below.



Thrombosis Journal 2004

Accumulating fibrin that polymerizes in the vessel wall triggers fibrinolysis. Fibrinolytic enzymes tissue plasminogen activating factor and urokinase plasminogen activating factor (tPA and uPA, respectively) are present in intima and are secreted by endothelial cells and likely play an important role in vascular remodeling [42-44]. Before addressing specific study findings a few general observations deserve attention.

- Strong associations between age and sex are found.
- D-dimer levels increase with age.

Other findings include:
- D-dimer levels are oftentimes associated with markers of inflammation
- Elevated levels of D-dimers are usually considered as a marker of increased clotting activity.

**User**: Find the specific location requiring changes to perform this request, 'Delete the first list item under in the second list'.

**System**: The first list item under the second list item can be found at [92, 398, 485, 459].

```
is_element('list_item',
                [92, 398, 485, 459])
check_element_order_from_top_or_title(
               'top', ‘list’,
               [92, 398, 485, 459], 1)
check_subelement_order_in_element(
              'list_item',
              [92, 398, 485, 459], 1)
```

Figure 3: Code-NLI based data cleaning. Generated data is treated as hypothesis, which is validated by LLM generated Python code, given PDF premise.

Given a lack of large-scale instruction-tuning data for grounding edit queries in PDFs, we build an automatic synthetic data generation process. We discuss below the two stages of this process:

**LLM powered data generation.** We present the PDF to an LLM and prompt it to generate edit requests along with the corresponding bounding box (see Figure 2). Prompt details in Appendix B. To ensure that the generated edit requests capture the compositional nature of real user queries, we structure PDFs in a hierarchical JSON format that explicitly defines the parent-child relationships of each element. Each element contains information about its type (e.g., paragraph, line, list), its bounding box within the PDF, and its subelements (e.g., a line is a subelement of a paragraph). Additionally, we organize elements sequentially within the same hierarchy based on their y-coordinates, enabling the LLM to generate edit requests that reference these elements numerically (e.g., "second list," "last line"). Moreover, the inclusion of a bounding box for each element allows the LLM to generate precise edit location in the system response for the corresponding edit query.

To obtain this hierarchical PDF representation, we require annotations for each PDF element, including its type, bounding box, and hierarchical subelements. However, there does not exist a single dataset that contains all this information for all element types. As such, we use different datasets – PubLayNet and DocLayNet (Pfitzmann et al., 2022) – each containing a subset of these elements. Table 1 shows a non-exhaustive list of these elements and the corresponding dataset containing them. If an element's hierarchical subelements' annotations do not exist natively in the dataset, we use heuristics to create them. These heuristics are relatively simple – comparing y-coordinates of words to create lines within paragraphs, comparing x-coordinates of lines to create list-items within lists, and so on. The granularity scope goes as low as words. This means we have annotations for words, equations, formulae, footnotes, page numbers, etc. Essentially, we tried to cover as many PDF elements as possible with the available datasets and heuristics. More details in Appendix A.

| | Title | Paragraph | | List | | | Tables | Figures |
|---|---|---|---|---|---|---|---|---|
| **Datasets** | | | Lines | | List-items | | | |
| | | | | | | Lines | | |
| PubLayNet | ✓ | ✓ | ♠ | ✓ | ♠ | ♠ | ✓ | ✓ |
| DocLayNet | ✓ | ✓ | ✓ | ♠ | ✓ | ✓ | ✓ | ✓ |

Table 1: Datasets and their PDF elements – either natively (✓) present or heuristically created (♠). Multiple datasets combine to give extensive elements coverage.

**Data cleaning with Code-NLI.** Since the LLM generates data automatically, it can be noisy—for instance, an edit request might reference the second list item while the bounding box corresponds to the third. To filter out such inconsistencies, we draw inspiration from Natural Language Inference (NLI) (Bowman et al., 2015), which predicts whether a hypothesis aligns with a given premise. Here, the premise is the PDF, and the hypothesis is the generated user-system conversation.

To verify these conversations, we generate sub-statements that must hold true. For example, to confirm that a bounding box corresponds to the second list item, a counting statement must evaluate its position as 2 (Figure 3). These sub-statements are generated by an LLM as Pythonic function calls, which are then executed using our implemented APIs for verification. Pythonic statements makes verification algorithmic and automatic. API details and prompts in Appendix C.

Once we have clean synthetic instruction-tuning data, we finetune a MLLM. The input is an image of the PDF and a user-edit request. The output is a bounding box localizing the edit element.

## 4 Experiments

To balance quality and the cost of long context, we use LLaMA-3.1 70B (Grattafiori and Team, 2024) for generating synthetic data. It is run on 40 Nvidia A100-80 GiB GPUs for 2 days. Next, GPT-4o (OpenAI, 2024) is used in Code-NLI cleaning as small context length allows us to optimize for quality. This results in ~3M samples for instruction tuning. We keep 95% samples for training and the rest

| Model | PDF A@0.5 | PDF A@0.30 | PDF A@0.25 | Design A@0.5 | Design A@0.30 | Design A@0.25 |
|---|---|---|---|---|---|---|
| *Zero-Shot* | | | | | | |
| GPT-4o | 5.624 | 14.258 | 17.885 | 28.961 | 46.766 | 52.582 |
| CogAgent | 4.16 | 9.42 | 11.13 | - | - | - |
| Qwen2.5-VL | 2.90 | 6.97 | 8.72 | - | - | - |
| Ferret-UI | 0.0 | 0.22 | 0.28 | - | - | - |
| Phi-3-V | 0.450 | 2.727 | 4.189 | 10.423 | 25.264 | 31.316 |
| DELOC | **14.703** | **30.391** | **35.198** | **32.421** | **51.489** | **57.445** |
| *Finetuned* | | | | | | |
| DocEdit | 36.500 | - | - | 34.340 | - | - |
| DocEdit-v2 | 48.690 | - | - | - | - | - |
| Phi-3-V | 26.567 | 48.524 | 54.371 | 34.523 | 57.183 | 62.272 |
| DELOC | **49.620** | **67.023** | **69.975** | **57.012** | **71.566** | **74.976** |

Table 2: Comparison of DELOC with state-of-the-art on DocEdit edit location (bbox) prediction. DELOC outperforms both existing Multimodal LLMs in zero-shot setting and specialized models in finetuned setting. $A@X$ denotes Accuracy at IoU of $X$.

| Model | A@0.5 |
|---|---|
| DELOC | 6.297 |
| – w/o CODE-NLI Filtering | 5.848 |

Table 3: Ablation of Code-NLI. The filtering step improves DELOC performance.

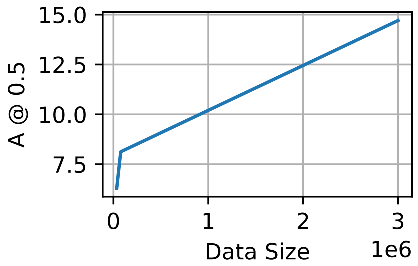


Figure 4: Datasize Ablation: DELOC's performance improves with data.

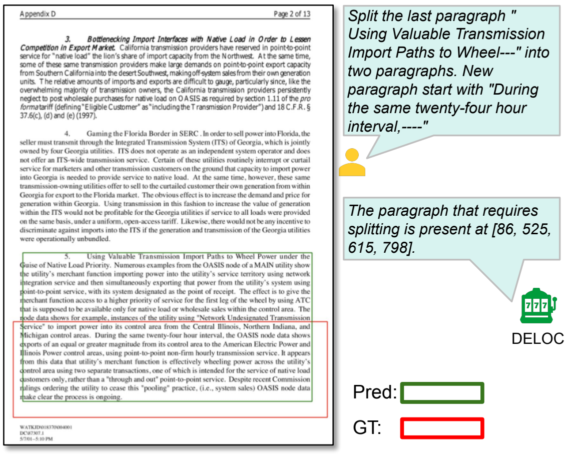

Split the last paragraph " Using Valuable Transmission Import Paths to Wheel---" into two paragraphs. New paragraph start with "During the same twenty-four hour interval,----"
The paragraph that requires splitting is present at [86, 525, 615, 798].
DELOC
Pred:
GT:

Figure 5: Qualitative sample prediction from DELOC.

for validation. More data statistics in Appendix D.

For instruction-tuning Phi-3V (Abdin and Team, 2024) is used as it's small and thus easy to train. We use a learning rate of 5e-5, weight decay of 0.12, AdamW optimizer, and a linear scheduler with a warmup. It takes approximatey 10 hours to train the model on 128 Nvidia A100-80 GiB GPUs. More architecture and training details are in Appendix E and Appendix F.

We evaluate our proposed approach on the test set of the DocEdit dataset (Mathur et al., 2023). The test is split into two subsets: PDFs, which are more verbose, and Designs, which are graphical. We use the originally proposed metric of calculating accuracy by evaluating if the Intersection Over Union (IoU) between predicted bbox and ground-truth bbox is greater than a threshold. For baselines, we select state-of-the-art MLLMs that understand text well from images. Consequently, we select GPT-4o, Phi3-V, CogAgent, Ferret-UI, Qwen2.5-VL (Bai et al., 2025) and Fuyu (Bavishi et al., 2023). We also consider as baselines specialized models – DocEdit v1 and v2 – that are finetuned on DocEdit. We compare DELOC against these baselines in two settings: zero-shot – where DELOC is not finetuned on the DocEdit train set, and finetuned – where it is.

Our main results are summarized in Table 2. We make the following observations: 1) DELOC outperforms both open-source and proprietary closed-source MLLM by ~3x on the PDF subset and ~12% on the Design subset. 2) DELOC outperforms all specialized finetuned models on both subsets. 3) DELOC improves over its backbone MLLM, Phi-3-V by 28x on PDFs and 3x on Designs, underscoring the significance of our synthetic instruction tuning. 4) The best existing zero-shot MLLM performance is at best 5%, indicating grounding in PDF is challenging. 5) All MLLM perform much better on the Design subset. This is understandable given all MLLM have been pretrained on natural images, which are closer in domain to Designs than to verbose PDFs. Notably, we found Fuyu to be quite bad; hence we do not include it in Table 2. We also don't compare against Agent-DocEdit as it includes an additional post-processing step that assumes availability of ground-truth bbox for all elements in a PDF.

Our ablation on Code-NLI in Table 3 indicates the importance of data cleaning to our approach. We also manually evaluate the quality of Code-NLI on 25 samples ourselves. Our findings are: 1) Precision is 84%: This implies that most of the data that Code-NLI predicts as correct is actually correct; only 16% incorrect data passes through the Code-NLI filter. 2) Recall is 63.64%: This implies it rejects a decent amount of good data (36%). This loss is acceptable for our use case since we generate a very large amount of synthetic data. The important consideration for us is that incorrect data should not be flagged as correct – which Code-NLI does (84% precision).

Figure 4 demonstrates that performance of DELOC improves with data size. We also provide a qualitative example in Figure 5 that shows 1) DELOC acquires effective capability of verbose PDF grounding. 2) Ambiguity in the expected response can lead to divergence from ground-truth. More examples in Appendix G.

## 5 Conclusion

In this work, we propose DELOC, a system to ground PDF edit request spatially. DELOC leverages spatial and hierarchical structure of PDFs to generate 3M synthetic instruction tuning data that successfully adapts existing MLLM for PDF grounding. Our results demonstrate that DELOC outperforms both existing MLLMs, including GPT-4o, and specialized models on DocEdit.

## Limitations

We attempted to cover a comprehensive list of PDF elements in our synthetically generated data. This coverage is mostly dependent on the element annotation present in PDF analysis datasets we began with to create our hierarchical representation. As such, there could be some PDF elements which are not covered. We leave for future work the addition of more PDF analysis datasets to increase element coverage. Furthermore, this method (like most grounding MLLM approaches) generates bounding boxes using auto-regressive prediction. The loss used assigns equal weight to all predictions not exactly same as ground-truth bbox, irrespective of their proximity to the ground-truth bbox. Ideally, the loss should be higher for a predicted bbox that is farther from the ground-truth than a predicted box that is closer to ground-truth. Reinforcement learning based preference optimization could be a way to address this issue in future works.

## References

Marah Abdin and The Phi-3 Team. 2024. Phi-3 technical report: A highly capable language model locally on your phone. *Preprint*, arXiv:2404.14219.

Shuai Bai, Keqin Chen, Xuejing Liu, Jialin Wang, Wenbin Ge, Sibo Song, Kai Dang, Peng Wang, Shijie Wang, Jun Tang, Humen Zhong, Yuanzhi Zhu, Mingkun Yang, Zhaohai Li, Jianqiang Wan, Pengfei Wang, Wei Ding, Zheren Fu, Yiheng Xu, Jiabo Ye, Xi Zhang, Tianbao Xie, Zesen Cheng, Hang Zhang, Zhibo Yang, Haiyang Xu, and Junyang Lin. 2025. Qwen2.5-vl technical report. *Preprint*, arXiv:2502.13923.

Rohan Bavishi, Erich Elsen, Curtis Hawthorne, Maxwell Nye, Augustus Odena, Arushi Somani, and Sağnak Taşırlar. 2023. Introducing our multimodal models.

Samuel R. Bowman, Gabor Angeli, Christopher Potts, and Christopher D. Manning. 2015. A large annotated corpus for learning natural language inference. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 632–642, Lisbon, Portugal. Association for Computational Linguistics.

Keqin Chen, Zhao Zhang, Weili Zeng, Richong Zhang, Feng Zhu, and Rui Zhao. 2023. Shikra: Unleashing multimodal llm's referential dialogue magic. *arXiv preprint arXiv:2306.15195*.

Xiaoyi Dong, Pan Zhang, Yuhang Zang, Yuhang Cao, Bin Wang, Linke Ouyang, Songyang Zhang, Haodong Duan, Wenwei Zhang, Yining Li, Hang Yan, Yang Gao, Zhe Chen, Xinyue Zhang, Wei Li, Jingwen Li, Wenhai Wang, Kai Chen, Conghui He, Xingcheng Zhang, Jifeng Dai, Yu Qiao, Dahua Lin, and Jiaqi Wang. 2024. Internlm-xcomposer2-4khd: A pioneering large vision-language model handling resolutions from 336 pixels to 4k hd. *Preprint*, arXiv:2404.06512.

Aaron Grattafiori and The LLaMA Team. 2024. The llama 3 herd of models. *Preprint*, arXiv:2407.21783.

Wenyi Hong, Weihan Wang, Qingsong Lv, Jiazheng Xu, Wenmeng Yu, Junhui Ji, Yan Wang, Zihan Wang, Yuxiao Dong, Ming Ding, and Jie Tang. 2023. Cogagent: A visual language model for gui agents. *Preprint*, arXiv:2312.08914.

Xin Lai, Zhuotao Tian, Yukang Chen, Yanwei Li, Yuhui Yuan, Shu Liu, and Jiaya Jia. 2024. Lisa: Reasoning segmentation via large language model. *Preprint*, arXiv:2308.00692.

Liunian Harold Li*, Pengchuan Zhang*, Haotian Zhang*, Jianwei Yang, Chunyuan Li, Yiwu Zhong, Lijuan Wang, Lu Yuan, Lei Zhang, Jenq-Neng Hwang, Kai-Wei Chang, and Jianfeng Gao. 2022. Grounded language-image pre-training. In *CVPR*.

Junpeng Liu, Tianyue Ou, Yifan Song, Yuxiao Qu, Wai Lam, Chenyan Xiong, Wenhu Chen, Graham Neubig, and Xiang Yue. 2024. Harnessing webpage uis for text-rich visual understanding. *Preprint*, arXiv:2410.13824.

Tengchao Lv, Yupan Huang, Jingye Chen, Lei Cui, Shuming Ma, Yaoyao Chang, Shaohan Huang, Wenhui Wang, Li Dong, Weiyao Luo, et al. 2023.

Kosmos-2.5: A multimodal literate model. *arXiv preprint arXiv:2309.11419*.

Chuofan Ma, Yi Jiang, Jiannan Wu, Zehuan Yuan, and Xiaojuan Qi. 2024. Groma: Localized visual tokenization for grounding multimodal large language models. *arXiv preprint arXiv:2404.13013*.

Puneet Mathur, Rajiv Jain, Jiuxiang Gu, Franck Dernoncourt, Dinesh Manocha, and Vlad Morariu. 2023. Docedit: Language-guided document editing. In *Thirty-Seventh AAAI Conference on Artificial Intelligence (AAAI)*.

OpenAI. 2024. Gpt-4o system card. *Preprint*, arXiv:2410.21276.

Birgit Pfitzmann, Christoph Auer, Michele Dolfi, Ahmed S. Nassar, and Peter Staar. 2022. Doclaynet: A large human-annotated dataset for document-layout segmentation. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, page 3743–3751. ACM.

Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, Gretchen Krueger, and Ilya Sutskever. 2021. Learning transferable visual models from natural language supervision. *Preprint*, arXiv:2103.00020.

Manan Suri, Puneet Mathur, Franck Dernoncourt, Rajiv Jain, Vlad I Morariu, Ramit Sawhney, Preslav Nakov, and Dinesh Manocha. 2024. DocEdit-v2: Document structure editing via multimodal LLM grounding. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 15485–15505, Miami, Florida, USA. Association for Computational Linguistics.

Weihan Wang, Qingsong Lv, Wenmeng Yu, Wenyi Hong, Ji Qi, Yan Wang, Junhui Ji, Zhuoyi Yang, Lei Zhao, Xixuan Song, Jiazheng Xu, Bin Xu, Juanzi Li, Yuxiao Dong, Ming Ding, and Jie Tang. 2023. Cogvlm: Visual expert for pretrained language models. *Preprint*, arXiv:2311.03079.

Te-Lin Wu, Rajiv Jain, Yufan Zhou, Puneet Mathur, and Vlad I Morariu. 2024. Agent-docedit: Language-instructed LLM agent for content-rich document editing. In *First Conference on Language Modeling*.

Haoxuan You, Haotian Zhang, Zhe Gan, Xianzhi Du, Bowen Zhang, Zirui Wang, Liangliang Cao, Shih-Fu Chang, and Yinfei Yang. 2023. Ferret: Refer and ground anything anywhere at any granularity. *Preprint*, arXiv:2310.07704.

Keen You, Haotian Zhang, Eldon Schoop, Floris Weers, Amanda Swearngin, Jeffrey Nichols, Yinfei Yang, and Zhe Gan. 2024. Ferret-ui: Grounded mobile ui understanding with multimodal llms. *Preprint*, arXiv:2404.05719.

Yuqian Yuan, Wentong Li, Jian Liu, Dongqi Tang, Xinjie Luo, Chi Qin, Lei Zhang, and Jianke Zhu. 2023. Osprey: Pixel understanding with visual instruction tuning. *Preprint*, arXiv:2312.10032.

Ao Zhang, Yuan Yao, Wei Ji, Zhiyuan Liu, and Tat-Seng Chua. 2023. Next-chat: An lmm for chat, detection and segmentation. *Preprint*, arXiv:2311.04498.

Xu Zhong, Jianbin Tang, and Antonio Jimeno Yepes. 2019. Publaynet: largest dataset ever for document layout analysis. *Preprint*, arXiv:1908.07836.

We provide additional details here for further clarification.

- Element Creation Heuristics (Appendix A)
- Synthetic Data Generation Prompt (Appendix B)
- Code-NLI Prompt and Samples (Appendix C)
- Data Statistics (Appendix D)
- Model Architecture (Appendix E)
- Additional Training Details (Appendix F)
- Additional Qualitative Examples (Appendix G)

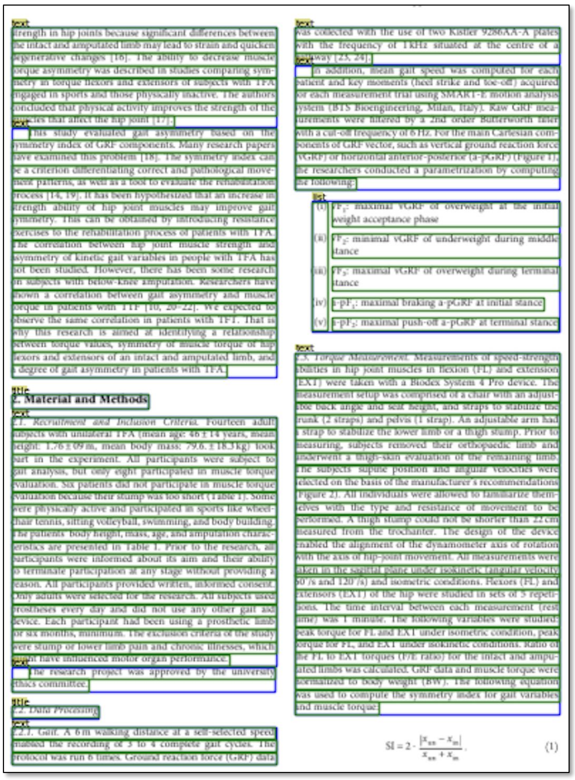


Figure 6: Sample output from using our heuristic to create lines and list-items.

## A Element Creation Heuristics

We require annotations for elements in PDFs to create the spatially-aware hierarchical representation. While an element annotation may be present in a dataset, its subelement or parent element may not have the requisite annotation available. We use heuristics to create annotations for such elements. To create lines within a paragraph, we take the bounding box of every word in the paragraph. All the words with same y-coordinates are considered to be in the same line. Similarly, to create lists from lines, we compare the x-coordinates of the beginning of the lines. Outliers are considered to be the beginning of a new list item. We illustrate the output from such heuristics to create lines and list-items in Figure 6. While these are simple, they work quite well.

## B Synthetic Data Generation Prompt

Once we have a hierarchical representation of the PDF elements along with their corresponding bounding box annotations, we feed them into an LLM to generate the instruction tuning data. Apart from the PDF elements, we input a system prompt containing the guidelines for generating an instruction tuning set and a one-shot example of how the user system conversation should be formatted. The process is illustrated in Figure 7.

The system prompt contains guidelines describing the task, input, and expected output. It contains instructions to generate diverse and compositional edit requests, to only utilize the information in the given hierarchical PDF representation, to not hallucinate, and so on. In addition, we provide a one-shot example to the LLM to further illustrate our requirements and guide it to follow expected format.

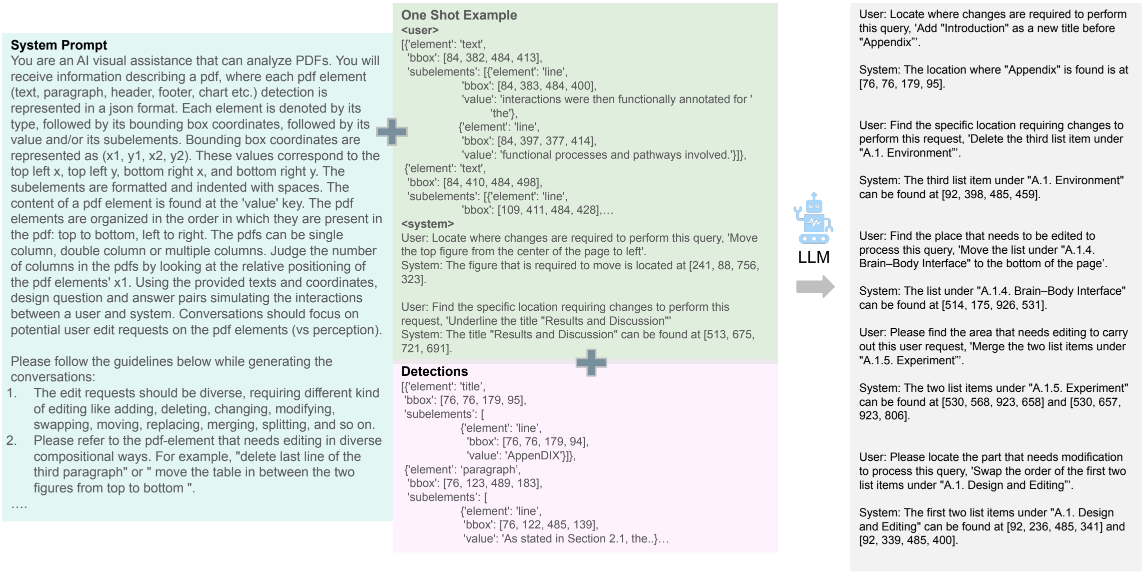
We provide the system prompt below:

```
You are an AI visual assistance that can analyze
     PDFs. You will receive information describing a
      pdf, where each pdf element (text, paragraph,
     header, footer, chart etc.) detection is
     represented in a json format. Each element is
     denoted by its type, followed by its bounding
     box coordinates, followed by its value and/or
     its subelements. Bounding box coordinates are
     represented as (x1, y1, x2, y2). These values
     correspond to the top left x, top left y,
     bottom right x, and bottom right y. The
     subelements are formatted and indented with
     spaces. The content of a pdf element is found
     at the 'value' key. The pdf elements are
     organized in the order in which they are
     present in the pdf: top to bottom, left to
     right. The pdfs can be single column, double
     column or multiple columns. Judge the number of
      columns in the pdfs by looking at the relative
      positioning of the pdf elements' x1.

Using the provided texts and coordinates, design
     question and answer pairs simulating the
     interactions between a user and system.
     Conversations should focus on potential user
     edit requests on the pdf elements (vs
     perception). Please follow the guidelines below
      while generating the conversations:

1. The edit requests should be diverse, requiring
     different kind of editing like adding, deleting
     , changing,modifying, swapping, moving,
     replacing, merging, splitting, and so on.

2. Please refer to the pdf-element that needs
     editing in diverse compositional ways. For
     example, \"delete last line of the third
```



**System Prompt**

You are an AI visual assistance that can analyze PDFs. You will receive information describing a pdf, where each pdf element (text, paragraph, header, footer, chart etc.) detection is represented in a json format. Each element is denoted by its type, followed by its bounding box coordinates, followed by its value and/or its subelements. Bounding box coordinates are represented as (x1, y1, x2, y2). These values correspond to the top left x, top left y, bottom right x, and bottom right y. The subelements are formatted and indented with spaces. The content of a pdf element is found at the 'value' key. The pdf elements are organized in the order in which they are present in the pdf: top to bottom, left to right. The pdfs can be single column, double column or multiple columns. Judge the number of columns in the pdfs by looking at the relative positioning of the pdf elements' x1. Using the provided texts and coordinates, design question and answer pairs simulating the interactions between a user and system. Conversations should focus on potential user edit requests on the pdf elements (vs perception).

Please follow the guidelines below while generating the conversations:

1. The edit requests should be diverse, requiring different kind of editing like adding, deleting, changing, modifying, swapping, moving, replacing, merging, splitting, and so on.
2. Please refer to the pdf-element that needs editing in diverse compositional ways. For example, "delete last line of the third paragraph" or " move the table in between the two figures from top to bottom ".

….

**One Shot Example**

**<user>**

```
[{'element': 'text',
  'bbox': [84, 382, 484, 413],
  'subelements': [{'element': 'line',
                 'bbox': [84, 383, 484, 400],
                 'value': 'interactions were then functionally annotated for '
                          'the'},
                {'element': 'line',
                 'bbox': [84, 397, 377, 414],
                 'value': 'functional processes and pathways involved.'}]},
 {'element': 'text',
  'bbox': [84, 410, 484, 498],
  'subelements': [{'element': 'line',
                 'bbox': [109, 411, 484, 428],…
```

**<system>**

User: Locate where changes are required to perform this query, 'Move the top figure from the center of the page to left'.
System: The figure that is required to move is located at [241, 88, 756, 323].

User: Find the specific location requiring changes to perform this request, 'Underline the title "Results and Discussion"'
System: The title "Results and Discussion" can be found at [513, 675, 721, 691].

**Detections**

```
[{'element': 'title’,
 'bbox': [76, 76, 179, 95],
 'subelements’: [
                {'element': 'line’,
                  'bbox': [76, 76, 179, 94],
                  'value': 'AppenDIX'}]},
 {'element': ‘paragraph’,
  'bbox': [76, 123, 489, 183],
  'subelements’: [
                {'element': 'line’,
                 'bbox': [76, 122, 485, 139],
                 'value': 'As stated in Section 2.1, the..}…
```

User: Locate where changes are required to perform this query, 'Add "Introduction" as a new title before "Appendix"’.

System: The location where "Appendix" is found is at [76, 76, 179, 95].

User: Find the specific location requiring changes to perform this request, 'Delete the third list item under "A.1. Environment"’.

System: The third list item under "A.1. Environment" can be found at [92, 398, 485, 459].

User: Find the place that needs to be edited to process this query, 'Move the list under "A.1.4. Brain–Body Interface" to the bottom of the page’.

System: The list under "A.1.4. Brain–Body Interface" can be found at [514, 175, 926, 531].

User: Please find the area that needs editing to carry out this user request, 'Merge the two list items under "A.1.5. Experiment"’.

System: The two list items under "A.1.5. Experiment" can be found at [530, 568, 923, 658] and [530, 657, 923, 806].

User: Please locate the part that needs modification to process this query, 'Swap the order of the first two list items under "A.1. Design and Editing"’.

System: The first two list items under "A.1. Design and Editing" can be found at [92, 236, 485, 341] and [92, 339, 485, 400].

Figure 7: Illustration of LLM prompting to generate synthetic instruction tuning data.

```
     paragraph\" or \" move the table in between the
      two figures from top to bottom \".

3. Please emphasize edit requests that deal with
     more abstract pdf elements rather low level
     elements like words and lines.

4. Use the x and y coordinates of the pdf-elements
     to figure out the relative spatial relations
     left of, right of,
top, bottom or to assess which paragraph is first,
     second or third.

5. Please count the number and order of pdf elements
      sequentially and correctly. Mistakes in
     counting are costly. Please think step by step
     while doing it.

6. Please do not generate edit requests that implies
      modifying multiple pdf elements when there
     exists actually only one, for example, asking
     to swap the order of list-items when acutally
     there is a single list item.

7. Please do not generate edits that call for left/
     right column, where there exists only a single
     column.

8. In the presence of list, please generate at least
      one edit requests on list bullets. The output
     then should also refer only to the bullets, not
      the whole list. Check to see whether the
     bullets are numeric, alphabetic, or plain
     circular bullets before requesting its editing.

9. Please geenrate the minimal bounding box that
     suffices to make the edit, i.e. ground the user
      request to the most fine-grained pdf-element.

10. Please generate multiple back and forth
     conversations. The user query should follow
     this format: '<phrase requesting the location>,
      \"<user edit query>\". For example: 'Please
     find the location to make this edit, \"Change
     list bullets to numeric\"'.

11. Only generate questions and answers that can be
     answered with the given information, do not use
      background information.

12. Please do not hallucinate pdf elements, its
     content, its subelements or edit requests.

13. It's better to not generate requests instead of
     generating wrong requests.
```

## C Code-NLI Prompt and Samples

To clean noisy synthetic data, we employe Code-NLI. The prompt used for Code-NLI includes a system prompt, that contains Python API function definitions, and few-shot prompts. We provide the system prompt given below:

```
You are an AI visual assistant that can do Natural
     Language Inference using python code. You will
     receive information describing a hypothesis.
     This includes a user edit request on pdf and a
     system output specifying the edit location with
      a bounding box. Your job is to create a python
      code, using listed APIs, that verifies that
     the bounding box correctly localizes the edit
     target in the user request. Each statement in
     the python code needs to be true to verify the
     correctness of the hypothesis.

The user edit request focuses on editing different
     type of pdf elements: title, paragraph, list,
     figure, and table. The elements could have
     subelements: title -> {bullets -> {bullet},
     list_item -> {line}}, paragraph -> {line},
     title -> {line}. It refers to different
     elements in a compositional and hierarchical
     manner. For example, last line of second
     paragraph, first list-item of third list, etc.
     The system output localizes the element/
     subelement requiring edit with a bounding box.
     The format of the bounding box is [x1,y1,x2,y2]
      where (x1,y1) is the coordinate of top left
     corner and (x2,y2) is bottom right corner,
     where x2>x1 and y2>y1.

Please use the following APIs to verify that the
     output bounding box indeed correctly captures
     the edit element/subelement. Please output only
      the python code using the listed APIs and
     nothing else. The element (dict) follows this
     format: {element_type: <element_type>, bbox: <[
     x1,y1,x2,y2]>, value: <value>, subelements:[<
     subelement1,..]}.

def is_element(element_type: str, bbox: list) ->
     bool:
      ""checks whether the element in bbox is
          element_type""
      ""Args:
          element_type (str): any element type '
               title', 'paragraph', 'list', 'line',
               'figure', 'table', 'bullets', '
```

```
            list_item', 'bullet'
        bbox (list): a bounding box of the type [
            x1, y1, x2, y2]
    ""

 def in_section(section_title: str, bbox: list) ->
     bool:
    ""checks whether the bbox is in the
        section_title""
    ""Args:
        section_title (str): the string contained
            in the title
        bbox (list): a bounding box of the type [
            x1, y1, x2, y2]
    ""

def contains_string(string: str, bbox: list) -> bool
    :
    ""checks whether the element in bbox contains
        string""
    ""Args:
        string (str): string to check
        bbox (list): a bounding box of the type [
            x1, y1, x2, y2]
    ""

def check_element_order_from_top_or_title(
    top_or_title (str), element_type: str, bbox:
    list, order: int, column: str = None) -> bool:
    ""checks the sequence order of the element in
        bbox of element_type from top of pdf or
        the section title where it lies""
    ""Args:
        top_or_title (str): Either 'top' or 'title
            ' from where to beign counting
        element_type (str): First hierarchy
            elements -- 'title', 'paragraph', '
            list', 'figure' or 'table'
        bbox (list): a bounding box of the type [
            x1, y1, x2, y2]
        order (int): denoting the sequence order
            of bbox of element_type. 1,2,3 order
            counts from top, -1,-2.. order counts
             from last
        column (str): to check the order in left
            or right column. Defaults to None
            when pdf is single column.
    ""

def check_subelement_order_from_top_or_title(
    top_or_title (str), subelement_type: str, bbox:
     list, order: int, column: str = None) -> int:
    ""checks the sequence order of the subelement
        in bbox of subelement_type from top of pdf
         or the section title where it lies""
    ""Args:
        top_or_title (str): Either 'top' or 'title
            ' from where to beign counting
        subelement_type (str): Second hierarchy
            elements -- 'line' (title and
            paragraph subelement), \
             'bullets' (list subelement), '
                 list_item' (list subelement)
        bbox (list): a bounding box of the type [
            x1, y1, x2, y2]
        order (int): denoting the sequence order
            of bbox of subelement_type. 1,2,3
            order counts from top, -1,-2.. order
            counts from last
        column (str): to check the order in left
            or right column. Defaults to None
            when pdf is single column.
    ""

def check_subelement_order_in_element(
    subelement_type: str, bbox: list, order: int)
    -> int:
    ""checks the sequence order of the subelement
        in bbox of subelement_type within the
        element in which bbox lies""
    ""Args:
        subelement_type (str): Second hierarchy
            elements -- 'line' (title and
            paragraph subelement), 'bullets' (
            list subelement), 'list_item' (list
            subelement)
        bbox (list): a bounding box of the type [
```

```
            x1, y1, x2, y2]
        order (int): denoting the sequence order
            of bbox of subelement_type. 1,2,3
            order counts from top, -1,-2.. order
            counts from last
    ""

def check_subsubelement_order_from_top_or_title(
    top_or_title (str), subsubelement_type: str,
    bbox: list, order: int, column: str = None) ->
    int:
    ""checks the sequence order of the
        subsubelement in bbox of
        subsubelement_type from top of pdf or the
        section title where it lies""
    ""Args:
        top_or_title (str): Either 'top' or 'title
            ' from where to beign counting
        subsubelement_type (str): Third hierarchy
            elements -- 'bullet' (subsubelement
            of bullets), 'line' (subsubelement of
             list_item)
        bbox (list): a bounding box of the type [
            x1, y1, x2, y2]
        order (int): denoting the sequence order
            of bbox of subelement_type. 1,2,3
            order counts from top, -1,-2.. order
            counts from last
        column (str): to check the order in left
            or right column. Defaults to None
            when pdf is single column.
    ""

def check_subsubelement_order_in_subelement(
    subsubelement_type: str, bbox: list, order: int
    ) -> int:
    ""checks the sequence order of the
        subsubelement in bbox of
        subsubelement_type within the subelement
        in which bbox lies""
     ""Args:
        subsubelement_type (str): Third hierarchy
            elements -- 'bullet' (subsubelement
            of bullets), 'line' (subsubelement of
             list_item)
        bbox (list): a bounding box of the type [
            x1, y1, x2, y2]
        order (int): denoting the sequence order
            of bbox of subsubelement_type. 1,2,3
            order counts from top, -1,-2.. order
            counts from last
    ""
```

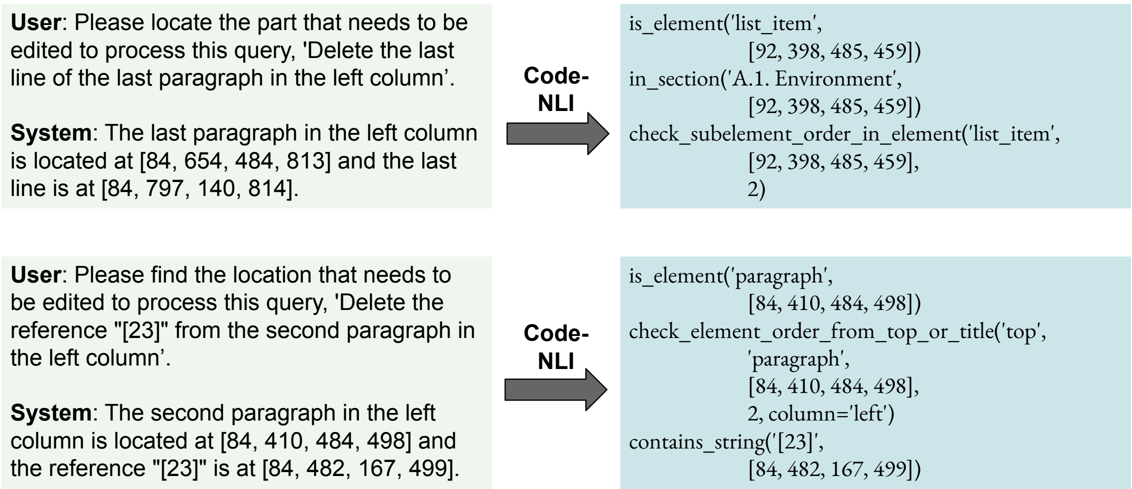
We also include some sample Python code generated by Code-NLI for verification in Figure 8.

## D Dataset Statistics

The total size of the synthetically generated data is 3M samples. The topic range is the same as the topic distribution of the base datasets: scientific articles, patents, finance, tenders, laws, and manuals. Similarly, image resolution is the same as base datasets: 1025 x 1025 for DocLayNet and 800x600 for PubLayNet.

## E Model Architecture

) Our model, DELOC, is based on Phi-3-V model. The Phi-3-V model's architecture is essentially composed of two components – an image encoder (CLIP ViT-L/14 (Radford et al., 2021)) and a transformer decoder (phi-3-mini). The visual tokens from the image encoder and concatenate with text



**User**: Please locate the part that needs to be edited to process this query, 'Delete the last line of the last paragraph in the left column'.

**System**: The last paragraph in the left column is located at [84, 654, 484, 813] and the last line is at [84, 797, 140, 814].

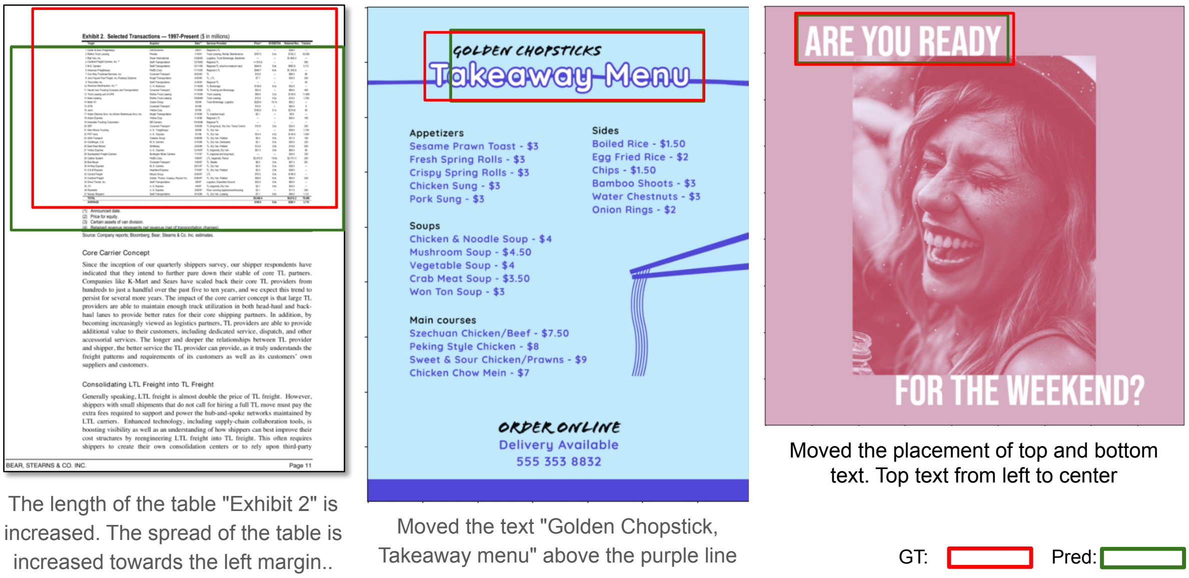
```
is_element('list_item',
            [92, 398, 485, 459])
in_section('A.1. Environment',
            [92, 398, 485, 459])
check_subelement_order_in_element('list_item',
            [92, 398, 485, 459],
            2)
```

**User**: Please find the location that needs to be edited to process this query, 'Delete the reference "[23]" from the second paragraph in the left column'.

**System**: The second paragraph in the left column is located at [84, 410, 484, 498] and the reference "[23]" is at [84, 482, 167, 499].

```
is_element('paragraph',
            [84, 410, 484, 498])
check_element_order_from_top_or_title('top',
            'paragraph',
            [84, 410, 484, 498],
            2, column='left')
contains_string('[23]',
            [84, 482, 167, 499])
```

Figure 8: Sample verification code generated by Code-NLI.



The length of the table "Exhibit 2" is increased. The spread of the table is increased towards the left margin..

Moved the text "Golden Chopstick, Takeaway menu" above the purple line

Moved the placement of top and bottom text. Top text from left to center

Figure 9: Additional Qualitative sample prediction from DELOC demonstrating its capabilities.

tokens in an interleaved manner and fed to the transformer decoder to output the prediction. In total, the number of parameters totals 4.2B.

The Phi-3-mini is a transformer decoder with 3072 hidden dimension, 32 heads and 32 layers.

## F Additional Training Details

The input image to our model is resized to 900x900. The text-heavy nature of PDF images requires that the model handle image resolutions dynamically according to the resolution of the text. Higher resolution text should result in higher image input resolution, and vice versa. This is taken care of by the dynamic cropping strategy (Dong et al., 2024) in the Phi-3-V model. It allows the input image to be dynamically cropped into a 2d array of blocks, which are then concatenated to represent the whole image.

## G Additional Qualitative Examples

We add more qualitative samples to further illustrate the capabilities of DELOC. As can be seen in Figure 9, DELOC is able to ground user edit requests well both for PDFs and design documents.