Supplementary Material for

SmolDocling: An ultra-compact vision-language model for end-to-end multi-modal document conversion

Ahmed Nassar   Matteo Omenetti   Maksym Lysak   Nikolaos Livathinos   Christoph Auer   Lucas Morin
Rafael Teixeira de Lima   Yusik Kim   A. Said Gurbuz   Michele Dolfi   Peter W. J. Staar

{ahn, mly, nli, cau, dol, taa}@zurich.ibm.com, {Matteo.Omenetti1}@ibm.com
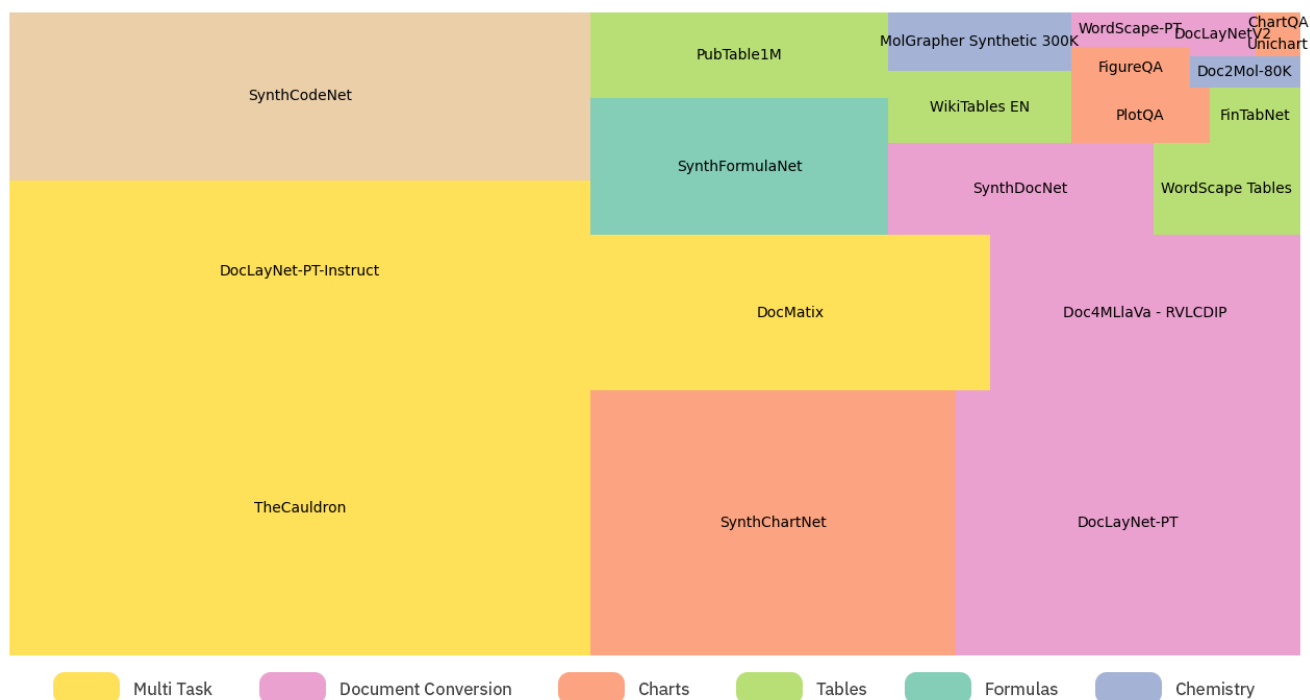
IBM Research

Figure 1. **Training datasets.** A treemap visualization of dataset sizes contributing to the training of SmolDocling, categorized by type. Each rectangle represents a dataset, with its area proportional to the total dataset size. Each color indicates a dataset type. *"Multi-Task"* refers to datasets that include multiple different tasks.

1. Additional Datasets Details

1.1. SynthChartNet

SynthChartNet includes four distinct types of charts: line, pie, bar, and stacked bar, generated using three visualization libraries: Matplotlib [**?**], Seaborn [**?**], and Pyecharts [**?**]. Each data set was rendered through all three libraries, producing a final dataset of 2.5 million visually diverse charts. The rendering process was designed to leverage the full range of features available in these libraries, ensuring visual variety while maintaining the plausibility of the data presented. Figure 2 depicts 10 samples from SynthChartNet.
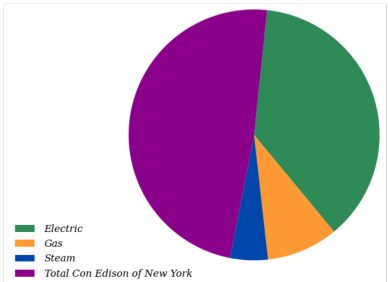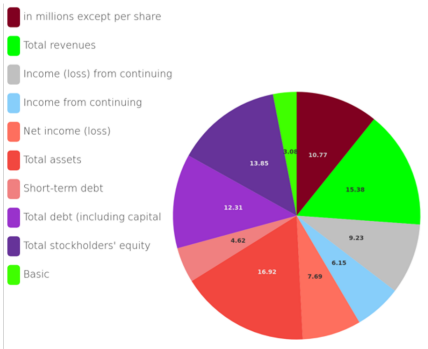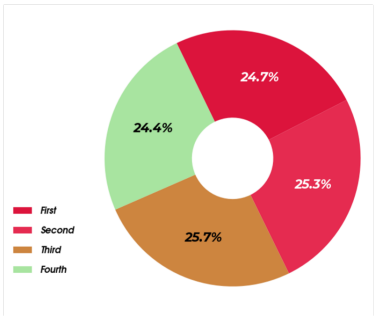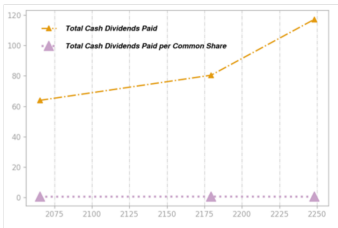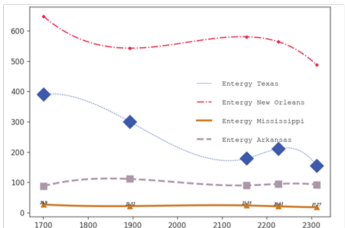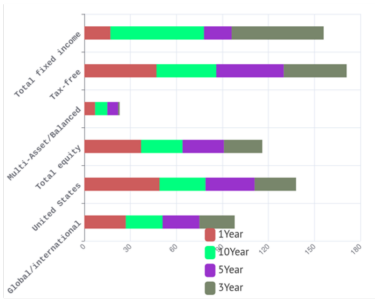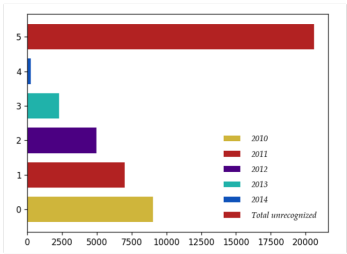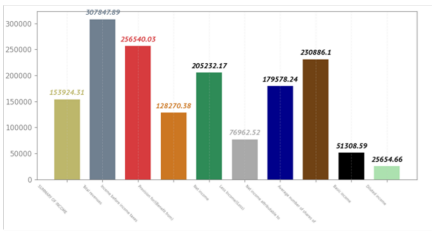
Figure 2. **Samples from SynthChartNet.**

Figure 3. **Samples from SynthCodeNet.**

1.2. SynthCodeNet

To render the code snippets of SynthCodeNet, we employed two rendering libraries: LaTeX and Pygments [**?**]. LaTex, in conjunction with its listings package, enabled the generation of code renderings similar to those commonly seen in academic and technical publications, while Pygments facilitated renderings with aesthetics resembling those of IDEs. The rendering process utilized the complete set of features provided by these libraries to enhance visual diversity while preserving the plausibility of the presented data. This included randomizing various stylistic parameters such as font, font size, style, weight, line spacing, tab size, background color, line numbering (including randomizing the starting line number), and programming language-specific syntax highlighting. Figure 3 shows 4 samples from SynthCodeNet.

$$\sum_{k\in J\cap K_m} e^{-i\varphi_k} m_{2Q_k}(e^{i\varphi_k}f)\nabla\chi_k = \sum_{k\in J\cap K_m}\left(e^{-i\varphi_k}m_{2Q_k}(e^{i\varphi_k}f)-e^{-i\bar{\varphi}_m}m_{2Q_k}(e^{i\bar{\varphi}_m}f)\right)\nabla\chi_k$$

$$+\sum_{k\in J\cap K_m} e^{-i\bar{\varphi}_m}\left(m_{2Q_k}(e^{i\bar{\varphi}_m}f)-m_{\tilde{Q}_m}(e^{i\bar{\varphi}_m}f)\right)\nabla\chi_k$$

$$+\sum_{k\in J\cap K_m} e^{-i\bar{\varphi}_m}m_{\tilde{Q}_m}(e^{i\bar{\varphi}_m}f)\nabla\chi_k \qquad \text{Y}$$

$$= I + II + III.$$

$$Rd \leq \alpha\ln\left(\frac{(A_y+\lambda_y)^{A_y+\lambda_y}}{(A_z+\lambda_z)^{A_z+\lambda_z}}\right)+(1-\alpha)\ln\left(\frac{\lambda_y^{\lambda_y}}{\lambda_z^{\lambda_z}}\right)-\ln\left(\frac{(A_y\alpha+\lambda_y)^{A_y\alpha+\lambda_y}}{(A_z\alpha+\lambda_z)^{A_z\alpha+\lambda_z}}\right) \qquad 4.30$$

$$R \leq \alpha\ln\left((A_y+\lambda_y)^{A_y+\lambda_y}\right)+(1-\alpha)\ln\left(\lambda_y^{\lambda_y}\right)-\ln\left((A_y\alpha+\lambda_y)^{A_y\alpha+\lambda_y}\right) \qquad 4.31$$

$$d \leq 1 \qquad 4.32$$

$$u_{ij}^{(0)} = \begin{cases} \frac{1}{(1+q)^{j-1}}, & (i=1), \\ \frac{q}{(1+q)^{j-i+1}}, & (2\leq i\leq j), \\ 0, & (j+1\leq i\leq N). \end{cases} \qquad \text{(L)}$$

$$d\Phi = \left(\frac{2\pi}{k}\right)^2 s_z^2 \tilde{a}(-\rho s)\tilde{a}^*(-\rho s)\, d\Omega.$$

$$\mathscr{L}_1 = \left\{\max(\frac{\bar{a}}{b}\gamma,\frac{\gamma(\bar{a}+b)-\sqrt{\gamma^2(\bar{a}+b)^2-4a\bar{b}}}{2\bar{b}})\leq\beta\leq\frac{\gamma(\bar{a}+b)+\sqrt{\gamma^2(\bar{a}+b)^2-4a\bar{b}}}{2\bar{b}}.\right\}$$

$$\mathscr{L}_2 = \left\{\frac{(a+\bar{b})+\sqrt{(a+\bar{b})^2-4\bar{a}b\gamma^2}}{2b\gamma}\leq\beta\leq\frac{\bar{a}}{b}\gamma\right\}$$

$$\mathscr{L}_3 = \left\{\beta\leq\min(\frac{\bar{a}}{b}\gamma,\frac{(a+\bar{b})-\sqrt{(a+\bar{b})^2-4\bar{a}b\gamma^2}}{2b\gamma})\right\}$$

$$\mathscr{L}_4 = \left\{\beta\leq\min(\frac{b\gamma}{a},\frac{\gamma(\bar{a}+b)-\sqrt{\gamma^2(\bar{a}+b)^2-4a\bar{b}}}{2a})\right\} \qquad \text{(x)}$$

$$\mathscr{L}_5 = \left\{\frac{\gamma(\bar{a}+b)+\sqrt{\gamma^2(\bar{a}+b)^2-4a\bar{b}}}{2a}\leq\beta\leq\frac{b\gamma}{a}\right\}$$

$$\mathscr{L}_6 = \left\{\max(\frac{b\gamma}{a},\frac{(a+\bar{b})-\sqrt{(a+\bar{b})^2-4\bar{a}b\gamma^2}}{2\bar{a}\gamma})\leq\beta\leq\frac{(a+\bar{b})+\sqrt{(a+\bar{b})^2-4\bar{a}b\gamma^2}}{2\bar{a}\gamma}\right\},$$

$$\int_{S^{n-1}} v(s)\, d\mu^\sigma \geq \frac{(n-2)\omega_{n-1}}{2}m^{U^\kappa b}$$
$$+\int_s^\infty\left(1-\frac{\cosh s}{\cosh r}\frac{u_0(r)}{u_0(s)}\right)\cosh r\sinh^{n-1} r\left(\int_{S^{n-1}} f(u(r,\theta))\, d\mu^\sigma\right) dr \qquad (21.64)$$

Figure 4. **Samples from SynthFormulaNet.**

1.3. SynthFormulaNet

The extracted formulas of SynthFormulaNet underwent a rigorous normalization procedure to ensure the model was trained only on correct and standardized code. The normalization process consisted of the following major steps:

1. **Token Filtering and Removal**: Unnecessary or redundant LaTeX tokens were removed or replaced based on predefined policies. This included:
   - Eliminating unwanted tokens.
   - Replacing tokens with equivalent ones for consistency.
2. **Structural Normalization**: The equation structure was standardized by:
   - Normalizing spacing between tokens.
   - Standardizing left and right delimiters.
   - Ensuring consistent use of tokens that require one or two braces.
3. **Token Simplification**: Redundant tokens and patterns were simplified, including:
   - Collapsing consecutive redundant tokens (e.g., repeated primes, dots, or excessive spacing).
   - Converting special constructs such as `\Big` into `\left` and `\right` for consistency.

The extracted equations were subsequently rendered using LaTex at 120 dpi. Several aspects of the rendering procedure have been randomized to ensure the visual diversity of the dataset. Each formula was rendered using a randomly selected font from a set of 85 possible options, with variations in font size, style, weight, and line spacing. Additionally, with a certain probability, each formula was enclosed within an "align" environment, allowing for the inclusion of a randomly generated equation number on either the left or right side of the equation. The equation number itself was chosen at random from a diverse set of possibilities, including small and large numbers, integers, floating-point values, or letters. Furthermore, it could be enclosed in single or double parentheses, brackets, or curly braces and was sometimes preceded by text labels such as "Eqn.", "Eqn", "Eq.", or "Equation". Figure 4 depicts 6 samples from SynthFormulaNet.

1.4. DocTags vocabulary

Here we provide a reference to a complete vocabulary of *DocTags* for *SmolDocling*, and their meaning.

We represent tags in an XML-like style of notation. Some tags have an opening and closing part, and wrap around textual content (or other tags) for example: `<text>hello world</text>`. Other tags are standalone and do not have a closing tag; they mark a special instruction, for example: `<page_break>`

The complete *DocTags* snippet can represent a page or multiple pages, wrapped into `<doctag>...</doctag>`. When content represents multiple pages, we use `<page_break>` tag as a separator.

Within the scope of `<doctag>` we place high-level, tags that wrap around the textual content of certain document blocks and identify the type of the block. These tags are: `<text>`, `<caption>`, `<footnote>`, `<formula>`, `<title>`, `<page_footer>`, `<page_header>`, `<picture>`, `<section_header>`, `<document_index>`, `<code>`, `<otsl>`, `<list_item>`, `<ordered_list>`, `<unordered_list>`

Each element can nest additional standalone location tags that encode its position on the page as a bounding box, represented in DocTags as `<loc_x1><loc_y1><loc_x2><loc_y2>`. Every x,y pair of coordinates belongs to a fixed grid. In our case we used integer values in the range from 0 to 500 which proportionally maps to width and height of the page.

`<otsl>` element contains table representation with following OTSL tags, that are standalone and used to describe structure of a table: `<fcel>`, `<ecel>`, `<lcel>`, `<ucel>`, `<xcel>`, `<nl>`. These tags follow rules as described in [**?**], we additionally differentiate OTSL notion of a cell into `<fcel>` - full cell, or cell which contain content, and `<ecel>` - empty cell. We also augment tabular structure information with extra tags `<ched>`, `<rhed>`, `<srow>` that replace `<fcel>` to describe cells which belong to column and row headers of the table, as well as table section accordingly. We mark such headers when ground truth about table headers from the underlying dataset is available.

`<list_item>` elements are placed within `<ordered_list>` or `<unordered_list>` and define weather it's enumerated (ordered) list or not.

`<picture>` and `<otsl>` elements by themselves can encapsulate `<caption>` tag if appropriate picture or table has it's own caption. This way we can connect extra descriptive information to illustrations and tables in the document.

`<code>` elements contain pieces of code as a content, and as such, respect tabulation and line breaks. Code elements also contains special standalone classification tag - `<_programming-language_>` where *programming-language* is an appropriate programming language name. Supported values (57 in total) for programming language names are: *Ada*, *Awk*, *Bash*, *bc*, *C*, *C#*, *C++*, *CMake*, *COBOL*, *CSS*, *Ceylon*, *Clojure*, *Crystal*, *Cuda*, *Cython*, *D*, *Dart*, *dc*, *Dockerfile*, *Elixir*, *Erlang*, *FORTRAN*, *Forth*, *Go*, *HTML*, *Haskell*, *Haxe*, *Java*, *JavaScript*, *Julia*, *Kotlin*, *Lisp*, *Lua*, *Matlab*, *MoonScript*, *Nim*, *OCaml*,

ObjectiveC, *Octave*, *PHP*, *Pascal*, *Perl*, *Prolog*, *Python*, *Racket*, *Ruby*, *Rust*, *SML*, *SQL*, *Scala*, *Scheme*, *Swift*, *TypeScript*, *unknown*, *VisualBasic*, *XML*, *YAML*

Within <picture> elements we include picture classification with extra standalone tags <*image_class*>. We classify images from our datasets into following categories: *natural_image*, *pie_chart*, *bar_chart*, *line_chart*, *flow_chart*, *scatter_chart*, *heatmap*, *remote_sensing*, *chemistry_molecular_structure*, *chemistry_markush_structure*, *icon*, *logo*, *signature*, *stamp*, *qr_code*, *bar_code*, *screenshot*, *map*, *stratigraphic_chart*, *cad_drawing*, *electrical_diagram*

2. Additional Details about the SmolDocling

Instruction	Description
Full conversion	Convert this page to docling.
Chart	Convert chart to table (e.g., <chart>).
Formula	Convert formula to LaTeX (e.g., <formula>).
Code	Convert code to text (e.g., <code>).
Table	Convert table to OTSL (e.g., <otsl>).
No-Code Actions/Pipelines	OCR the text in a specific location: <loc_155><loc_233><loc_206><loc_237>
	Identify element at: <loc_247><loc_482><loc_252><loc_486>
	Find all 'text' elements on the page, retrieve all section headers.
	Detect footer elements on the page.

Table 1. **Supported Conversion Instructions.**
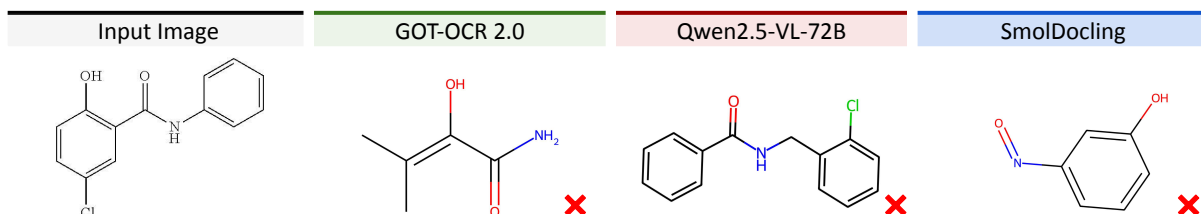
Figure 5. **Qualitative comparison for molecule image recognition.** Predictions are shown for GOT-OCR 2.0, Qwen2.5-VL-72B and SmolDocling on a simple molecule image.

3. Additional Qualitative Analysis

3.1. Molecule Image Recognition Experiment

Extracting molecule images from documents has potential to accelerate research in chemistry and materials discovery [**?**]. Some recent document understanding models, such as GOT-OCR 2.0 [**?**] and Qwen2.5-VL [**?**], claim to perform molecule image recognition in documents. Following this direction, we experimented with training SmolDocling for molecule image recognition on full document pages and cropped images. In order to predict molecule structures (graphs) with SmolDocling, we use the molecule string identifier named SMILES [**?**].

Datasets. For training on cropped images, we use the MolGrapher-Synthetic-300K dataset [**?**]. The dataset contains 300K synthetic samples. It is created using real chemical-structures retrieved from the database PubChem [**?**] which are then synthetically drawn using the library RDKit [**?**]. For training of full pages, we use Doc2Mol-80K, a subset of the PatCID dataset [**?**]. Our subset contains 80 000 real patent documents published in the United-States between 2022 and 2024.

Evaluation. Figure 5 shows an example of predicted molecules for a simple molecule image. While some models capture high-level molecular features, they consistently fail to reconstruct its detailed structure. Notably, this molecule is trivial for specialized models such as MolGrapher [**?**], MolScribe [**?**] or DECIMER [**?**], which accurately recognize its structure. Reducing the gap between general document understanding models and specialized models may require incorporating a specialized vocabulary for SMILES tokens, explicitly encoding atoms and bonds, and extending the SMILES sequence with localization tokens for each atom.

3.2. Layout analysis samples

To illustrate and compare the visual grounding capabilities of SmolDocling, a set of sample predictions with SmolDocling and Qwen2.5-VL from DocLayNet [**?**] is shown in Table 2. Note that the element location results are independent from the correct reproduction of document content and structure.

#	Ground-Truth	SmolDocling	Qwen2.5-VL	ID
1				0b49b1c3d521741a7c9b92d960d5f61
2				0aafe8d4b66ac7ab81e9456327fbdb01

#	Ground-Truth	SmolDocling	Qwen2.5-VL	ID
3				1bff94fbe34e976d53eabc5b8b43f355
4				2ab06e940ef25ff7b2e832ab8176de15
5				ebd1b976af12cd1d16d58939049920df

#	Ground-Truth	SmolDocling	Qwen2.5-VL	ID
6				eae40317d83ba98d3e70427e6d7baff6

Table 2. Visualizations of layout output from SmolDocling and QwenVL-2.5 compared to the DocLayNet ground truth. Examples are chosen to be representative of different layout styles and features. The prediction results however do not represent a generalizable measure of the model's performance on inputs with similar features. (1) Multi-column pages are handled by SmolDocling and Qwen2.5-VL, with some recall errors in the latter. (2) A manual page with terminal output shows poor bounding box recall on SmolDocling and label confusion in Qwen2.5-VL. (3) Lists with nesting are handled well in SmolDocling but confuse Qwen2.5-VL. (4) Both SmolDocling and Qwen2.5-VL reconstruct equations well, however with different annotation conventions on including or excluding the equation index. (5) On a portrait page with colorful elements and gradient background, SmolDocling creates less accurate bounding-boxes and Qwen2.5-VL suffers low recall. (6) On a report page with tables and diagrams, SmolDocling output exhibits some repetition loop, fabricating non-existent text cells (bottom left), while Qwen2.5-VL is confused between tables and pictures.

References