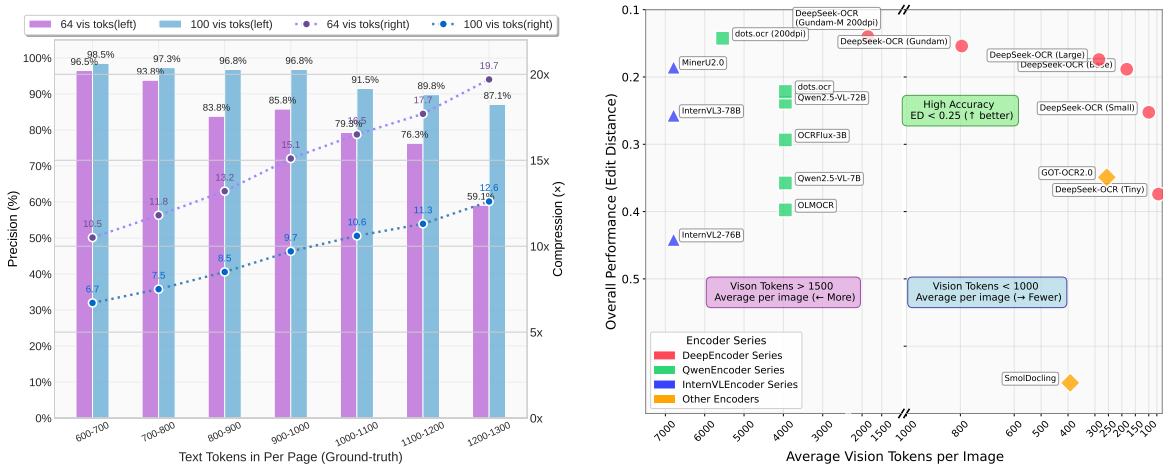
DeepSeek-OCR: Contexts Optical Compression

Haoran Wei, Yaofeng Sun, Yukun Li

DeepSeek-AI

Abstract

We present DeepSeek-OCR as an initial investigation into the feasibility of compressing long contexts via optical 2D mapping. DeepSeek-OCR consists of two components: DeepEncoder and DeepSeek3B-MoE-A570M as the decoder. Specifically, DeepEncoder serves as the core engine, designed to maintain low activations under high-resolution input while achieving high compression ratios to ensure an optimal and manageable number of vision tokens. Experiments show that when the number of text tokens is within 10 times that of vision tokens (i.e., a compression ratio < 10×), the model can achieve decoding (OCR) precision of 97%. Even at a compression ratio of 20×, the OCR accuracy still remains at about 60%. This shows considerable promise for research areas such as historical long-context compression and memory forgetting mechanisms in LLMs. Beyond this, DeepSeek-OCR also demonstrates high practical value. On OmniDocBench, it surpasses GOT-OCR2.0 (256 tokens/page) using only 100 vision tokens, and outperforms MinerU2.0 (6000+ tokens per page on average) while utilizing fewer than 800 vision tokens. In production, DeepSeek-OCR can generate training data for LLMs/VLMs at a scale of 200k+ pages per day (a single A100-40G). Codes and model weights are publicly accessible at http://github.com/deepseek-ai/DeepSeek-OCR.



(a) Compression on Fox benchmark

(b) Performance on Omnidocbench

Figure 1 | Figure (a) shows the compression ratio (number of text tokens in ground truth/number of vision tokens model used) testing on Fox [21] benchmark; Figure (b) shows performance comparisons on OmniDocBench [27]. DeepSeek-OCR can achieve state-of-the-art performance among end-to-end models enjoying the fewest vision tokens.

Contents

1. Introduction

Current Large Language Models (LLMs) face significant computational challenges when processing long textual content due to quadratic scaling with sequence length. We explore a potential solution: leveraging visual modality as an efficient compression medium for textual information. A single image containing document text can represent rich information using substantially fewer tokens than the equivalent digital text, suggesting that optical compression through vision tokens could achieve much higher compression ratios.

This insight motivates us to reexamine vision-language models (VLMs) from an LLM-centric perspective, focusing on how vision encoders can enhance LLMs' efficiency in processing textual information rather than basic VQA [12, 16, 24, 32, 41] what humans excel at. OCR tasks, as an intermediate modality bridging vision and language, provide an ideal testbed for this vision-text compression paradigm, as they establish a natural compression-decompression mapping between visual and textual representations while offering quantitative evaluation metrics.

Accordingly, we present DeepSeek-OCR, a VLM designed as a preliminary proof-of-concept for efficient vision-text compression. Our work makes three primary contributions:

First, we provide comprehensive quantitative analysis of vision-text token compression ratios. Our method achieves 96%+ OCR decoding precision at 9-10× text compression, ~90% at 10-12× compression, and ~60% at 20× compression on Fox [21] benchmarks featuring diverse document layouts (with actual accuracy being even higher when accounting for formatting differences between output and ground truth), as shown in Figure 1(a). The results demonstrate that compact language models can effectively learn to decode compressed visual representations, suggesting that larger LLMs could readily acquire similar capabilities through appropriate pretraining design.

Second, we introduce DeepEncoder, a novel architecture that maintains low activation memory and minimal vision tokens even with high-resolution inputs. It serially connects window attention and global attention encoder components through a 16× convolutional compressor. This design ensures that the window attention component processes a large number of vision tokens, while the compressor reduces vision tokens before they enter the dense global attention component, achieving effective memory and token compression.

Third, we develop DeepSeek-OCR based on DeepEncoder and DeepSeek3B-MoE [19, 20]. As shown in Figure 1(b), it achieves state-of-the-art performance within end-to-end models on OmniDocBench while using the fewest vision tokens. Additionally, we equip the model with capabilities for parsing charts, chemical formulas, simple geometric figures, and natural images to enhance its practical utility further. In production, DeepSeek-OCR can generate 33 million pages of data per day for LLMs or VLMs using 20 nodes (each with 8 A100-40G GPUs).

In summary, this work presents a preliminary exploration of using visual modality as an efficient compression medium for textual information processing in LLMs. Through DeepSeek-OCR, we demonstrate that vision-text compression can achieve significant token reduction (7-20×) for different historical context stages, offering a promising direction for addressing long-context challenges in large language models. Our quantitative analysis provides empirical guidelines for VLM token allocation optimization, while the proposed DeepEncoder architecture showcases practical feasibility with real-world deployment capabilities. Although focused on OCR as a proof-of-concept, this paradigm opens new possibilities for rethinking how vision and language modalities can be synergistically combined to enhance computational efficiency in large-scale text processing and agent systems.

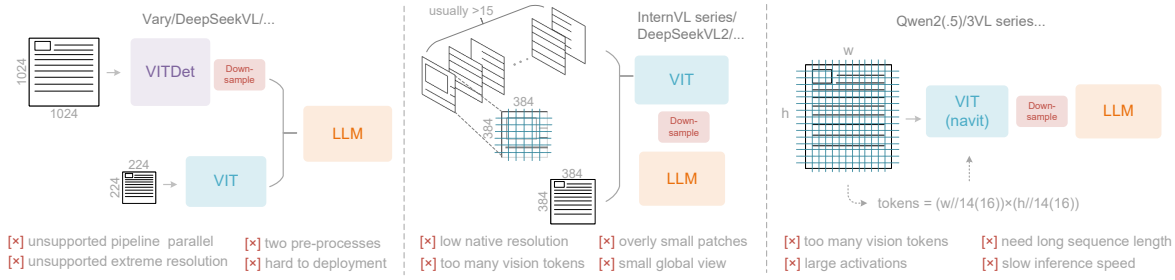


Figure 2 | Typical vision encoders in popular VLMs. Here are three types of encoders commonly used in current open-source VLMs, all of which suffer from their respective deficiencies.

2. Related Works

2.1. Typical Vision Encoders in VLMs

Current open-source VLMs employ three main types of vision encoders, as illustrated in Figure 2. The first type is a dual-tower architecture represented by Vary [36], which utilizes parallel SAM [17] encoder to increase visual vocabulary parameters for high-resolution image processing. While offering controllable parameters and activation memory, this approach suffers from significant drawbacks: it requires dual image preprocessing that complicates deployment and makes encoder pipeline parallelism challenging during training. The second type is tile-based method exemplified by InternVL2.0 [8], which processes images by dividing them into small tiles for parallel computation, reducing activation memory under high-resolution settings. Although capable of handling extremely high resolutions, this approach has notable limitations due to its typically low native encoder resolution (below 512×512), causing large images to be excessively fragmented and resulting in numerous vision tokens. The third type is adaptive resolution encoding represented by Qwen2-VL [35], which adopts the NaViT [10] paradigm to directly process full images through patch-based segmentation without tile parallelization. While this encoder can handle diverse resolutions flexibly, it faces substantial challenges with large images due to massive activation memory consumption that can cause GPU memory overflow, and sequence packing requires extremely long sequence lengths during training. Long vision tokens will slow down both prefill and generation phases of inference.

2.2. End-to-end OCR Models

OCR, particularly document parsing task, has been a highly active topic in the image-to-text domain. With the advancement of VLMs, a large number of end-to-end OCR models have emerged, fundamentally transforming the traditional pipeline architecture (which required separate detection and recognition expert models) by simplifying OCR systems. Nougat [6] first employs end-to-end framework for academic paper OCR on arXiv, demonstrating the potential of models in handling dense perception tasks. GOT-OCR2.0 [38] expands the scope of OCR2.0 to include more synthetic image parsing tasks and designs an OCR model with performance-efficiency trade-offs, further highlighting the potential of end-to-end OCR researches. Additionally, general vision models such as Qwen-VL series [35], InternVL series [8], and many their derivatives continuously enhance their document OCR capabilities to explore dense visual perception boundaries. However, a crucial research question that current models have not addressed is: *for a document containing 1000 words, how many vision tokens are at least needed for decoding?* This question holds significant importance for research in the principle that "*a picture is worth a thousand words.*"

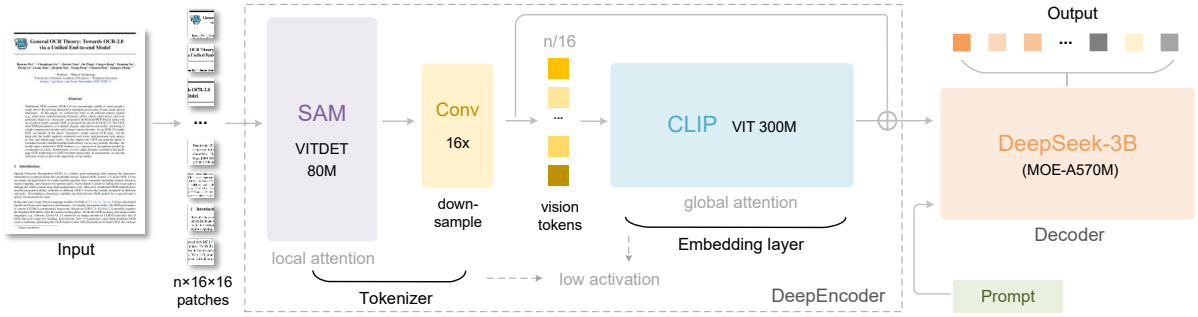


Figure 3 | The architecture of DeepSeek-OCR. DeepSeek-OCR consists of a DeepEncoder and a DeepSeek-3B-MoE decoder. DeepEncoder is the core of DeepSeek-OCR, comprising three components: a SAM [17] for perception dominated by window attention, a CLIP [29] for knowledge with dense global attention, and a 16× token compressor that bridges between them.

3. Methodology

3.1. Architecture

As shown in Figure 3, DeepSeek-OCR enjoys a unified end-to-end VLM architecture consisting of an encoder and a decoder. The encoder (namely DeepEncoder) is responsible for extracting image features and tokenizing as well as compressing visual representations. The decoder is used for generating the required result based on image tokens and prompts. DeepEncoder is approximately 380M in parameters, mainly composed of an 80M SAM-base [17] and a 300M CLIP-large [29] connected in series. The decoder adopts a 3B MoE [19, 20] architecture with 570M activated parameters. In the following paragraphs, we will delve into the model components, data engineering, and training skills.

3.2. DeepEncoder

To explore the feasibility of contexts optical compression, we need a vision encoder with the following features: 1.Capable of processing high resolutions; 2.Low activation at high resolutions; 3.Few vision tokens; 4.Support for multiple resolution inputs; 5. Moderate parameter count. However, as described in the Section 2.1, current open-source encoders cannot fully satisfy all these conditions. Therefore, we design a novel vision encoder ourselves, named DeepEncoder.

3.2.1. Architecture of DeepEncoder

DeepEncoder mainly consists of two components: a visual perception feature extraction component dominated by window attention, and a visual knowledge feature extraction component with dense global attention. To benefit from the pretraining gains of previous works, we use SAM-base (patch-size 16) and CLIP-large as the main architectures for the two components respectively. For CLIP, we remove the first patch embedding layer since its input is no longer images but output tokens from the previous pipeline. Between the two components, we borrow from Vary [36] and use a 2-layer convolutional module to perform 16× downsampling of vision tokens. Each convolutional layer has a kernel size of 3, stride of 2, padding of 1, and channels increase from 256 to 1024. Assuming we input a 1024×1024 image, the DeepEncoder will segment it into 1024/16×1024/16=4096 patch tokens. Since the first half of encoder is dominated by window attention and only 80M, the activation is acceptable. Before entering global attention,

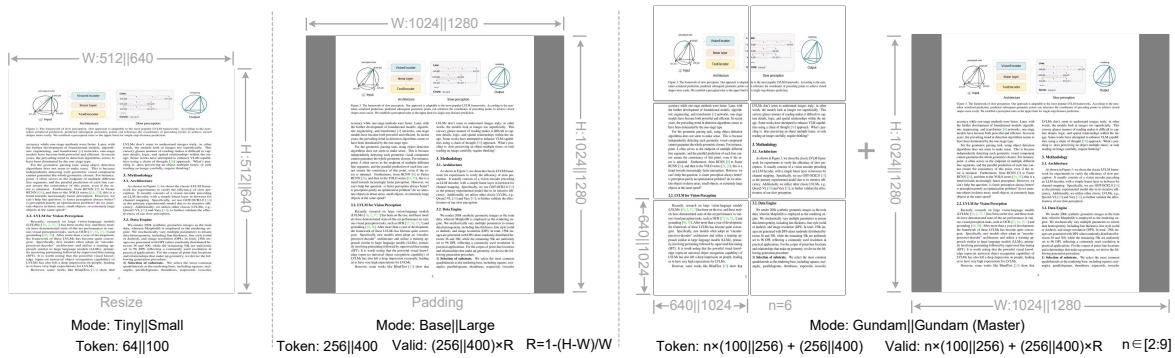


Figure 4 | To test model performance under different compression ratios (requiring different numbers of vision tokens) and enhance the practicality of DeepSeek-OCR, we configure it with multiple resolution modes.

the 4096 tokens go through the compression module and the token count becomes 4096/16=256, thus making the overall activation memory controllable.

Table 1 | Multi resolution support of DeepEncoder. For both research and application purposes, we design DeepEncoder with diverse native resolution and dynamic resolution modes.

Mode	Native Resolution				Dynamic Resolution	
	Tiny	Small	Base	Large	Gundam	Gundam-M
Resolution	512	640	1024	1280	640+1024	1024+1280
Tokens	64	100	256	400	n×100+256	n×256+400
Process	resize	resize	padding	padding	resize + padding	resize + padding

3.2.2. Multiple resolution support

Suppose we have an image with 1000 optical characters and we want to test how many vision tokens are needed for decoding. This requires the model to support a variable number of vision tokens. That is to say the DeepEncoder needs to support multiple resolutions.

We meet the requirement aforementioned through dynamic interpolation of positional encodings, and design several resolution modes for simultaneous model training to achieve the capability of a single DeepSeek-OCR model supporting multiple resolutions. As shown in Figure 4, DeepEncoder mainly supports two major input modes: native resolution and dynamic resolution. Each of them contains multiple sub-modes.

Native resolution supports four sub-modes: Tiny, Small, Base, and Large, with corresponding resolutions and token counts of 512×512 (64), 640×640 (100), 1024×1024 (256), and 1280×1280 (400) respectively. Since Tiny and Small modes have relatively small resolutions, to avoid wasting vision tokens, images are processed by directly resizing the original shape. For Base and Large modes, in order to preserve the original image aspect ratio, images are padded to the corresponding size. After padding, the number of valid vision tokens is less than the actual number of vision tokens, with the calculation formula being:

$$N_{valid} = \lceil N_{actual} \times [1 - ((max(w,h) - min(w,h))/(max(w,h)))] \rceil \quad (1)$$

where w and h represent the width and height of the original input image.

Dynamic resolution can be composed of two native resolutions. For example, Gundam mode consists of n×640×640 tiles (local views) and a 1024×1024 global view. The tiling method following InternVL2.0 [8]. Supporting dynamic resolution is mainly for application considerations, especially for ultra-high-resolution inputs (such as newspaper images). Tiling is a form of secondary window attention that can effectively reduce activation memory further. It's worth noting that due to our relatively large native resolutions, images won't be fragmented too much under dynamic resolution (the number of tiles is controlled within the range of 2 to 9). The vision token number output by the DeepEncoder under Gundam mode is: $n \times 100 + 256$, where n is the number of tiles. For images with both width and height smaller than 640, n is set to 0, i.e., Gundam mode will degrade to Base mode.

Gundam mode is trained together with the four native resolution modes to achieve the goal of one model supporting multiple resolutions. Note that Gundam-master mode (1024×1024 local views+1280×1280 global view) is obtained through continued training on a trained DeepSeek-OCR model. This is mainly for load balancing, as Gundam-master's resolution is too large and training it together would slow down the overall training speed.

3.3. The MoE Decoder

Our decoder uses the DeepSeekMoE [19, 20], specifically DeepSeek-3B-MoE. During inference, the model activates 6 out of 64 routed experts and 2 shared experts, with about 570M activated parameters. The 3B DeepSeekMoE is very suitable for domain-centric (OCR for us) VLM research, as it obtains the expressive capability of a 3B model while enjoying the inference efficiency of a 500M small model.

The decoder reconstructs the original text representation from the compressed latent vision tokens of DeepEncoder as:

$$f_{\text{dec}} : \mathbb{R}^{n \times d_{\text{latent}}} \rightarrow \mathbb{R}^{N \times d_{\text{text}}}; \quad \hat{\mathbf{X}} = f_{\text{dec}}(\mathbf{Z}) \quad \text{where } n \leq N \tag{2}$$

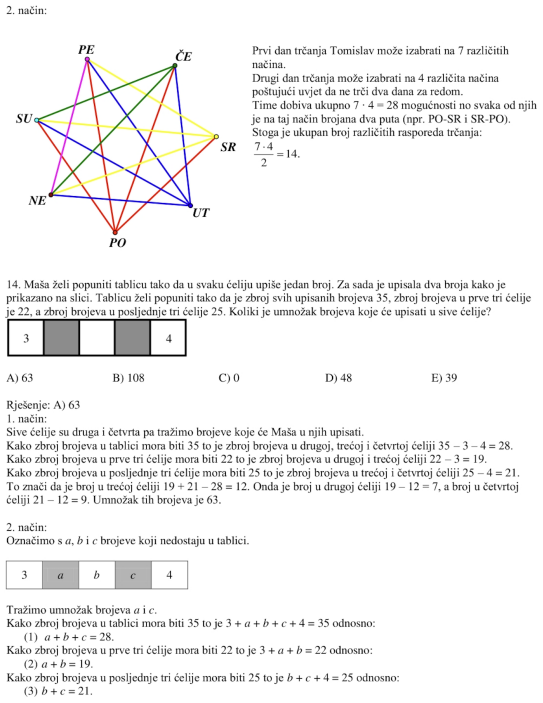
where $\mathbf{Z} \in \mathbb{R}^{n \times d_{\text{latent}}}$ are the compressed latent(vision) tokens from DeepEncoder and $\hat{\mathbf{X}} \in \mathbb{R}^{N \times d_{\text{text}}}$ is the reconstructed text representation. The function f_{dec} represents a non-linear mapping that can be effectively learned by compact language models through OCR-style training. It is reasonable to conjecture that LLMs, through specialized pretraining optimization, would demonstrate more natural integration of such capabilities.

3.4. Data Engine

We constructe complex and diverse training data for DeepSeek-OCR, including OCR 1.0 data, which mainly consists of traditional OCR tasks such as scene image OCR and document OCR; OCR 2.0 data, which mainly includes parsing tasks for complex artificial images, such as common charts, chemical formulas, and plane geometry parsing data; General vision data, which is mainly used to inject certain general image understanding capabilities into DeepSeek-OCR and preserve the general vision interface.

3.4.1. OCR 1.0 data

Document data is the top priority for DeepSeek-OCR. We collect 30M pages of diverse PDF data covering about 100 languages from the Internet, with Chinese and English accounting for approximately 25M and other languages accounting for 5M. For this data, we create two types of ground truth: coarse annotations and fine annotations. Coarse annotations are extracted



2. način:

Prvi dan trčanja Tomislav može izabrati na 7 različitih načina.
Drugi dan trčanja može izabrati na 4 različita načina poštujući uvjet da ne trči dva dana za redom.
Time dobiva ukupno $7 \cdot 4 = 28$ mogućnosti no svaka od njih je na taj način brojana dva puta (npr. PO-SR i SR-PO).
Stoga je ukupan broj različitih rasporeda trčanja:
$\frac{7 \cdot 4}{2} = 14.$

14. Maša želi popuniti tablicu tako da u svaku ćeliju upiše jedan broj. Za sada je upisala dva broja kako je prikazano na slici. Tablicu želi popuniti tako da je zbroj svih upisanih brojeva 35, zbroj brojeva u prve tri ćelije je 22, a zbroj brojeva u posljednje tri ćelije 25. Koliki je umnožak brojeva koje će upisati u sive ćelije?

3				4

A) 63 B) 108 C) 0 D) 48 E) 39

Rješenje: A) 63
1. način:
Sive ćelije su druga i četvrta pa tražimo brojeve koje će Maša u njih upisati.
Kako zbroj brojeva u tablici mora biti 35 to je zbroj brojeva u drugoj, trećoj i četvrtoj ćeliji $35 - 3 - 4 = 28$.
Kako zbroj brojeva u prve tri ćelije mora biti 22 to je zbroj brojeva u drugoj i trećoj ćeliji $22 - 3 = 19$.
Kako zbroj brojeva u posljednje tri ćelije mora biti 25 to je zbroj brojeva u trećoj i četvrtoj ćeliji $25 - 4 = 21$.
To znači da je broj u trećoj ćeliji $19 + 21 - 28 = 12$. Onda je broj u drugoj ćeliji $19 - 12 = 7$, a broj u četvrtoj ćeliji $21 - 12 = 9$. Umnožak tih brojeva je 63.

2. način:
Označimo s a, b i c brojeve koji nedostaju u tablici.

3	a	b	c	4

Tražimo umnožak brojeva a i c.
Kako zbroj brojeva u tablici mora biti 35 to je $3 + a + b + c + 4 = 35$ odnosno:
(1) $a + b + c = 28$.
Kako zbroj brojeva u prve tri ćelije mora biti 22 to je $3 + a + b = 22$ odnosno:
(2) $a + b = 19$.
Kako zbroj brojeva u posljednje tri ćelije mora biti 25 to je $b + c + 4 = 25$ odnosno:
(3) $b + c = 21$.

(a) Ground truth image

```
<|ref|>text<|/ref|><|det|>[[55, 43, 130, 60]]<|/det|>
2. način:

<|ref|>image<|/ref|><|det|>[[70, 93, 450, 360]]<|/det|>


<|ref|>text<|/ref|><|det|>[[460, 95, 896, 132]]<|/det|>
Prvi dan trčanja Tomislav može izabrati na 7 različitih načina.

<|ref|>text<|/ref|><|det|>[[460, 131, 880, 168]]<|/det|>
Drugi dan trčanja može izabrati na 4 različita načina poštujući uvjet da ne trči
dva dana za redom.

<|ref|>text<|/ref|><|det|>[[460, 166, 941, 220]]<|/det|>
Time dobiva ukupno \(7 \cdot 4 = 28\) mogućnosti no svaka od njih je na taj način
brojana dva puta (npr. PO-SR i SR-PO). Stoga je ukupan broj različitih rasporeda
trčanja:

<|ref|>equation<|/ref|><|det|>[[460, 217, 550, 256]]<|/det|>
\[ \frac{7 \cdot 4}{2} = 14. \]

<|ref|>text<|/ref|><|det|>[[55, 397, 931, 452]]<|/det|>
14. Maša želi popuniti tablicu tako da u svaku ćeliju upiše jedan broj. Za sada
je upisala dva broja kako je prikazano na slici. Tablicu želi popuniti tako da je
zbroj svih upisanih brojeva 35, zbroj brojeva u prve tri ćelije je 22, a zbroj
brojeva u posljednje tri ćelije 25. Koliki je umnožak brojeva koje će upisati u
sive ćelije?

<|ref|>table<|/ref|><|det|>[[57, 450, 360, 500]]<|/det|>
<table><tr><td>3</td><td></td><td></td><td></td><td>4</td></tr></table>

<|ref|>text<|/ref|><|det|>[[55, 515, 110, 534]]<|/det|>
A) 63

<|ref|>text<|/ref|><|det|>[[230, 515, 293, 534]]<|/det|>
B) 108

<|ref|>text<|/ref|><|det|>[[405, 515, 450, 534]]<|/det|>
C) 0

<|ref|>text<|/ref|><|det|>[[581, 515, 636, 534]]<|/det|>
D) 48
```

(b) Fine annotations with layouts

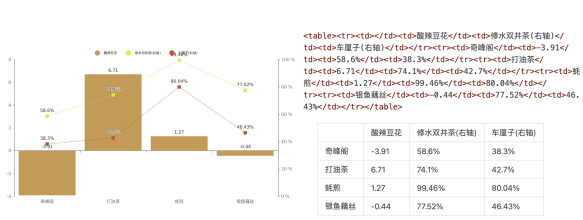
Figure 5 | OCR 1.0 fine annotations display. We format the ground truth into an interleaved layout and text format, where each paragraph of text is preceded by the coordinates and label of it in the original image. All coordinates are normalized into 1000 bins.

directly from the full dataset using *fitz*, aimed at teaching the model to recognize optical text, especially in minority languages. Fine annotations include 2M pages each for Chinese and English, labeled using advanced layout models (such as PP-DocLayout [33]) and OCR models (such as MinuerU [34] and GOT-OCR2.0 [38]) to construct detection and recognition interleaved data. For minority languages, in the detection part, we find that the layout model enjoys certain generalization capabilities. In the recognition part, we use *fitz* to create small patch data to train a GOT-OCR2.0, then use the trained model to label small patches after layout processing, employing a model flywheel to create 600K data samples. During the training of DeepSeek-OCR, coarse labels and fine labels are distinguished using different prompts. The ground truth for fine annotation image-text pairs can be seen in Figure 5. We also collect 3M *Word* data, constructing high-quality image-text pairs without layout by directly extracting content. This data mainly brings benefits to formulas and HTML-formatted tables. Additionally, we select some open-source data [28, 37] as supplements.

For natural scene OCR, our model mainly supports Chinese and English. The image data sources come from LAION [31] and Wukong [13], labeled using PaddleOCR [9], with 10M data samples each for Chinese and English. Like document OCR, natural scene OCR can also control whether to output detection boxes through prompts.

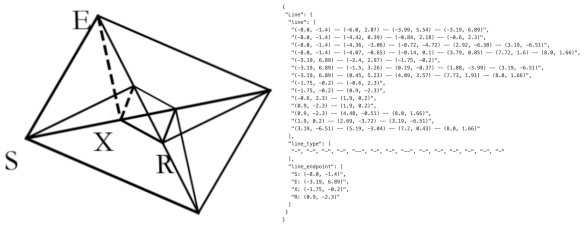
3.4.2. *OCR 2.0 data*

Following GOT-OCR2.0 [38], we refer to chart, chemical formula, and plane geometry parsing data as OCR 2.0 data. For chart data, following OneChart [7], we use pyecharts and matplotlib



<table><tr><td></td><td>酸辣豆花</td><td>修水双井茶(右轴)</td><td>车厘子(右轴)</td></tr><tr><td>奇峰阁</td><td>-3.91</td><td>58.6%</td><td>38.3%</td></tr><tr><td>打油茶</td><td>6.71</td><td>74.1%</td><td>42.7%</td></tr><tr><td>鱿鹏</td><td>1.27</td><td>99.46%</td><td>80.04%</td></tr><tr><td>银鱼藏丝</td><td>-0.44</td><td>77.52%</td><td>46.43%</td></tr></table>

(a) Image-text ground truth of chart



(b) Image-text ground truth of geometry

Figure 6 | For charts, we do not use OneChart's [7] dictionary format, but instead use HTML table format as labels, which can save a certain amount of tokens. For plane geometry, we convert the ground truth to dictionary format, where the dictionary contains keys such as line segments, endpoint coordinates, line segment types, etc., for better readability. Each line segment is encoded using the Slow Perception [39] manner.

to render 10M images, mainly including commonly used line, bar, pie, and composite charts. We define chart parsing as image-to-HTML-table conversion task, as shown in Figure 6(a). For chemical formulas, we utilize SMILES format from PubChem as the data source and render them into images using RDKit, constructing 5M image-text pairs. For plane geometry images, we follow Slow Perception [39] for generation. Specifically, we use perception-ruler size as 4 to model each line segment. To increase the diversity of rendered data, we introduce geometric translation-invariant data augmentation, where the same geometric image is translated in the original image, corresponding to the same ground truth drawn at the centered position in the coordinate system. Based on this, we construct a total of 1M plane geometry parsing data, as illustrated in Figure 6(b).

3.4.3. General vision data

DeepEncoder can benefit from CLIP's pretraining gains and has sufficient parameters to incorporate general visual knowledge. Therefore, we also prepare some corresponding data for DeepSeek-OCR. Following DeepSeek-VL2 [40], we generate relevant data for tasks such as caption, detection, and grounding. Note that DeepSeek-OCR is not a general VLM model, and this portion of data accounts for only 20% of the total data. We introduce such type of data mainly to preserve the general vision interface, so that researchers interested in our model and general vision task can conveniently advance their work in the future.

3.4.4. Text-only data

To ensure the model's language capabilities, we introduced 10% of in-house text-only pretrain data, with all data processed to a length of 8192 tokens, which is also the sequence length for DeepSeek-OCR. In summary, when training DeepSeek-OCR, OCR data accounts for 70%, general vision data accounts for 20%, and text-only data accounts for 10%.

3.5. Training Pipelines

Our training pipeline is very simple and consists mainly of two stages: a).Training DeepEncoder independently; b).Training the DeepSeek-OCR. Note that the Gundam-master mode is obtained by continuing training on a pre-trained DeepSeek-OCR model with 6M sampled data. Since the training protocol is identical to other modes, we omit the detailed description hereafter.

3.5.1. Training DeepEncoder

Following Vary [36], we utilize a compact language model [15] and use the next token prediction framework to train DeepEncoder. In this stage, we use all OCR 1.0 and 2.0 data aforementioned, as well as 100M general data sampled from the LAION [31] dataset. All data is trained for 2 epochs with a batch size of 1280, using the AdamW [23] optimizer with cosine annealing scheduler [22] and a learning rate of 5e-5. The training sequence length is 4096.

3.5.2. Training DeepSeek-OCR

After DeepEncoder is ready, we use data mentioned in Section 3.4 to train the DeepSeek-OCR. with the entire training process conducted on the HAI-LLM [14] platform. The entire model uses pipeline parallelism (PP) and is divided into 4 parts, with DeepEncoder taking two parts and the decoder taking two parts. For DeepEncoder, we treat SAM and the compressor as the vision tokenizer, place them in PP0 and freeze their parameters, while treating the CLIP part as input embedding layer and place it in PP1 with unfrozen weights for training. For the language model part, since DeepSeek3B-MoE has 12 layers, we place 6 layers each on PP2 and PP3. We use 20 nodes (each with 8 A100-40G GPUs) for training, with a data parallelism (DP) of 40 and a global batch size of 640. We use the AdamW optimizer with a step-based scheduler and an initial learning rate of 3e-5. For text-only data, the training speed is 90B tokens/day, while for multimodal data, the training speed is 70B tokens/day.

Table 2 | We test DeepSeek-OCR's vision-text compression ratio using all English documents with 600-1300 tokens from the Fox [21] benchmarks. Text tokens represent the number of tokens after tokenizing the ground truth text using DeepSeek-OCR's tokenizer. Vision Tokens=64 or 100 respectively represent the number of vision tokens output by DeepEncoder after resizing input images to 512×512 and 640×640.

Text Tokens	Vision Tokens =64		Vision Tokens=100		
	Precision	Compression	Precision	Compression	Pages
600-700	96.5%	10.5×	98.5%	6.7×	7
700-800	93.8%	11.8×	97.3%	7.5×	28
800-900	83.8%	13.2×	96.8%	8.5×	28
900-1000	85.9%	15.1×	96.8%	9.7×	14
1000-1100	79.3%	16.5×	91.5%	10.6×	11
1100-1200	76.4%	17.7×	89.8%	11.3×	8
1200-1300	59.1%	19.7×	87.1%	12.6×	4

4. Evaluation

4.1. Vision-text Compression Study

We select Fox [21] benchmarks to verify DeepSeek-OCR's compression-decompression capability for text-rich documents, in order to preliminarily explore the feasibility and boundaries of contexts optical compression. We use the English document portion of Fox, tokenize the ground truth text with DeepSeek-OCR's tokenizer (vocabulary size of approximately 129k), and select documents with 600-1300 tokens for testing, which happens to be 100 pages. Since the number of text tokens is not large, we only need to test performance in Tiny and Small modes, where Tiny mode corresponds to 64 tokens and Small mode corresponds to 100 tokens. We use the prompt