視覚言語モデルを利用した漢籍用 OCR 実現の試み

守岡 知彦 *

1　はじめに

深層学習技術を用いたいわゆる AI OCR の登場以降、従来、OCR（Optical character recognition; 光学文字認識）での認識が非常に困難であった崩字・草書で書かれた日本の古典文献を実用的な精度で機械的読み取りが可能となった。特に、NDL 古典籍 OCR (ver.3) では、そのソースコードやデータセット、実行環境の Docker イメージが、自由なライセンス (CC BY 4.0) で公開されたことからさまざまな利用者が崩字用 OCR として利用することとなった。

NDL 古典籍 OCR (ver.3) は崩字で書かれた古典日本語（古語）のテキストの読み取り読み取りを得意とする反面、一見それより容易そうな楷書で書かれた漢籍の読み取りは不得意である。特に割注を含む漢籍の版本の場合、人手での校正を前提としたとしても、あまり役に立たず、全部一から入れ直した方が良いということになりがちである。これは NDL 古典籍 OCR は古語を含む日本語の読み取りのためにチューンされていて、漢文（古典中国語）のための学習データをあまり持っていないことに起因すると考えられる。

ところで、2025 年 10 月に DeepSeek-OCR [1) 2)] が公開され注目を集めた。これは視覚言語モデル (Vision-Language Model; VLM) を利用した OCR, というよりは画像理解のための言語モデルを利用した OCR であり、特に、DeepSeek-OCR は「視覚情報の圧縮」というアプローチを用いて、視覚情報をその構造（や意味）を壊さずに圧縮することでモデルのサイズを縮小し少ない計算リソースでも現実的な速度で実行可能とした点が注目を集めた。また、DeepSeek-OCR は、学習に用いたデータセット自体は非公開であるものの、実行に必要となるデータセットやプログラムを自由なライセンス（MIT ライセンス）で公開していることも研究用の基盤として注目に値する。そして、DeepSeek-OCR は中国の DeepSeek 社の大規模言語モデルをベースにしていると考えられることから大量の漢籍の

* 国文学研究資料館

1) https://deepseek-ocr.io/jp/

2) `https://github.com/deepseek-ai/DeepSeek-OCR`

コーパスを学習しているのではないかと考えられ、漢籍用 OCR として有望ではないかと考えられた。

同じ頃、中国のアリババ社も大規模言語モデル Qwen（通義千問）シリーズの視覚言語モデル Qwen-VL の最新版として Qwen3-VL シリーズを相次いで公開した。Qwen3-VL では DeepStack における視覚特徴量を複数の層に注入するアプローチを採用しており、OCR 性能を強化するとともにハルシネーションを抑制するように工夫されている。Qwen-VL も大量の漢籍のコーパスから学習していると考えられるのと DeepSeek-OCR よりも重いものの絶対性能では勝ると考えられるため、こちらも漢籍用 OCR として有望ではないかと考えられた。

そこで、今回は実際に DeepSeek-OCR と Qwen3-VL を用いて古典中国語の版本用の OCR を構成し、テキスト化の実験を行った。

2 実行環境

現実的に OCR を実行可能な視覚言語モデル (VLM) は計算量が多くモデルサイズも大きいため、大量のメモリーを搭載した GPU が必要となる。DeepSeek-OCR の場合、Base モード（解像度 1024×1024, 平均ビジョン・トークン 256）であれば GPU メモリー 12GB の NVIDIA GeForce RTX 3060 でも動作するが、Large モード（解像度 1280×1280, 平均ビジョン・トークン 400）や Gundam モード（解像度 $n \times 640 + 1024$）や Gundam-M モード（解像度 $n \times 1024 + 1280$）の場合、大容量メモリーを搭載した強力な GPU が必要となる。

大容量メモリーを搭載した強力な GPU はとても効果であるが、NVIDIA 製の GPU よりは性能は劣るものの、比較的低価格で大容量メモリーを搭載した GPU を利用するためのソリューションが今年に入り複数登場している。

一つは Apple シリコンを搭載した Mac で、これ自体は数年前から登場しているが、その登場時点では未整備だった機械学習用フレームワークも徐々に整備が進み、アップルが開発した MLX を利用することで比較的簡単に VLM を含む LLM が実行可能となった。ハードウェア的には、Mac は CPU と GPU とでメモリーを共有するアーキテクチャであるが、他の同種のアーキテクチャーに比べてメモリーの転送速度が高いため、CPU から GPU への転送が律速となる問題での性能が出やすいといえる。また、MLX は統一メモリモデルを持っており、データを移動せずに CPU と GPU のどちらでも実行可能となっている。ユーザーから見た場合、CPU と GPU のメモリー割り当て量を再起動することなく OS 上で動的に変更できる（実際には、OS 任せにして利用者は意識しなくても良い）ため使い勝手が良い。2025 年 12 月現在、アカデミック割引価格を利用した場合、MacBook Pro に 128GB のメモリーを搭載した場合で 70 万円弱、Mac Studio に 512GB の

メモリーを搭載した場合で 1351300 円である。

２つ目は今年の秋に登場した NVIDIA DGX Spark である。これは ARM64 アーキテクチャーの CPU と NVIDIA 製 GPU で 128GB のメモリーを共有する小型ワークステーションで、メモリーの転送速度は遅いものの、CUDA 用のソフトウェアがそのまま動くという特徴がある。各社から製品が出ており、2025 年 12 月現在、70 万円前後のようである。

３つ目は AMD Ryzen Al Max+ 395 (Strix Halo) を搭載したミニ PC である。これは AMD64 (x86-64) アーキテクチャーの一般的な CPU と AMD の GPU (Radeon 8060S; gfx1151) を一つのチップにまとめた APU (Accelerated Processing Unit) に CPU と GPU で共有するメモリーを搭載するソリューションで、メモリーの転送速度は遅いものの、比較的安価に大容量 GPU メモリーを搭載した環境が構築できる。AI 用のソフトウェアやモデルの多くは CUDA を前提に書かれているが、AMD GPU 用のプラットフォームである ROCm の CUDA 互換性が向上し、特に、ROCm 7 になってからは実用性が向上したことから大規模言語モデルの実行環境として実用的になった。また、NVIDIA と異なり、AMD の GPU ではソフトウェアスタックをオープンソースにしているため、将来的に Debian 等の OS に標準装備される可能性が高く、NVIDIA GPU を用いた CUDA 環境に比べ取り回しが良くなる可能性がある。但し、現在、gfx1151 で ROCm 7 を利用するためには開発中のものを入れる必要があり、AMD のドキュメントの記述も混乱していることから環境構築のための情報の入手性に難があると言える。なお、2025 年 12 月現在、128GB のメモリーを搭載するミニ PC が 45 万円前後で購入でき、GPU メモリー 96GB の環境を作る場合、最も安価であるといえる。

今回は、

- AMD Ryzen Al Max+ 395 (CPU 16 コア、32 スレッド、GPU: Radeon 8060S (gfx1151)) を搭載したメモリー 128GB のミニ PC（Minisforum MS-S1 MAX)
- Mac mini 2024（Apple M4 Pro (CPU 14 コア（パフォーマンス 10、高効率 4)、GPU 20 コア、メモリー 64GB）
- MacBook Pro 2023 (Apple M3 Pro (CPU 11 コア（パフォーマンス 5、高効率 6)、GPU 14 コア、メモリー 36GB）

を用いた。

2.1 Radeon 8060S (gfx1151) 用環境の構築

Debian 13 (trixie) 上で、`https://instinct.docs.amd.com/projects/amdgpu-docs/en/latest/install/detailed-install/package-manager/package-manager-debian.html` を参考にカーネルドライバーを導入した後、`https://community.frame.work/t/`

compiling-vllm-from-source-on-strix-halo/77241 を参考に ROCm 7.10.0 上に vLLM 環境を構築した。

2.2 Mac 用環境の構築

Apple シリコンを搭載した Mac の場合、MLX-VLM [3] を入れることで視覚言語モデルが利用できる。これは

```
pip install -U mlx-vlm
```

とすることで入れられ、依存関係が少なくインストールが比較的簡単である。

但し、OS 標準の /usr/bin/python3 (Python 3.9.6) の場合、インストールされるのは v0.1.15 であるが、Qwen3-VL が動くのは v0.3.4 以降である。

そこで、Homebrew [4] を使って、

```
% brew install uv
```

のように Python 用パッケージマネージャー uv をインスールし、

```
% uv venv py3.13-mlx-vlm -p 3.13
```

のように Python 3.13 + MLX-VLM 用の仮想環境を作成し、

```
% source py3.13-mlx-vlm/bin/activate
```

を実行してこの仮想環境に入る。ここで、

```
(py3.13-mlx-vlm) % uv run python --version
```

を実行すると

```
Python 3.13.11
```

のように Python 3.13 が入っていることが確認できる。

なお、この環境から出る場合、

```
(py3.13-mlx-vlm) % deactivate
```

を実行すれば良い。また、再びこの仮想環境に入る場合、

```
% source py3.13-mlx-vlm/bin/activate
```

を実行すれば良い。ここで、シェルのプロンプト部

```
(py3.13-mlx-vlm) %
```

は仮想環境 py3.13-mlx-vlm にいることを示し、シェルのプロンプト部

```
%
```

は仮想環境の外にいることを示す。実際に入力するのは % より後の部分である。

この Python 3.13 用の環境 py3.13-mlx-vlm で、

3) https://github.com/Blaizzy/mlx-vlm
4) https://brew.sh/

```
(py3.13-mlx-vlm) % uv pip install -U mlx-vlm
```

を実行することでこの環境に MLX-VLM をインストールすることができる。

インストールした MLX-VLM のバージョンは

```
(py3.13-mlx-vlm) % uv pip show mlx-vlm
```

で調べることができる。上記を実行したとき、

```
Using Python 3.13.11 environment at: .
Name: mlx-vlm
Version: 0.3.9
        ⋮
```

のようになっていれば 2025 年 12 月現在の MLX-VLM の最新版である v0.3.9 が入っていることが確認できる。

3 DeepSeek-OCR

ここでは Apple シリコンを搭載した Mac 上で MLX-VLM を用いて実行する例を示す：

```
(py3.13-mlx-vlm) % uv run python -m mlx_vlm.generate \
  --model mlx-community/DeepSeek-OCR-8bit \
  --max-tokens 128 \
  --temperature 0.0 \
  --prompt "Output every lines." \
  --image /tmp/kokusho.nijl.ac.jp/api/iiif/200018997/v4/NIKI/096-0005/096-0005-
00005.tif/full/pct:50/0/default.jpg
```

これを実行すると

```
<frozen runpy>:128: RuntimeWarning: 'mlx_vlm.generate' found in sys.modules after \
import of package 'mlx_vlm', but prior to execution of 'mlx_vlm.generate'; this may \
result in unpredictable behaviour
Calling `python -m mlx_vlm.generate ...` directly is deprecated. \
Use `mlx_vlm generate` or `python -m mlx_vlm generate` instead.
Fetching 13 files: 100%|████████████████████████████████████████████████
████████████████████████████████████████████████████████████████████████
████████████| 13/13 [00:00<00:00, 301248.35it/s]
Add pad token = ['<｜▁pad▁｜>'] to the tokenizer
<｜▁pad▁｜>:2
Add image token = ['<image>'] to the tokenizer
<image>:128815
Added grounding-related tokens
Added chat tokens
==========
Files: ['/tmp/kokusho.nijl.ac.jp/api/iiif/200018997/v4/NIKI/096-0005/096-0005-
```

```
00005.tif/full/pct:50/0/default.jpg']

Prompt: <image>
Output every lines.
====================
BASE:  (1, 256, 1280)
PATCHES:  (6, 100, 1280)
====================
三神矣乾道獨化所必成此純男
一書曰天地初判一物在於虛中執難言
書曰天地初判一物在於虛中執難言
立尊次國秩次尊次國次立尊次國次
立尊次國秩次尊次國次立尊次國秩次
尊次國秩次尊次國秩次尊次國秩次
尊次國秩次尊次國秩次尊次國秩次
尊次國秩次尊次國秩次尊次國秩次
尊次
==========
Prompt: 899 tokens, 518.315 tokens-per-sec
Generation: 128 tokens, 60.190 tokens-per-sec
Peak memory: 5.380 GB
```

のような結果が出力される。比較的高速で、古典中国語を理解しているような挙動を示すが、理解できなくなるとハルシネーションを起こす。その際、画像中に出てきた文字列を延々と繰り返すような挙動を示す。

プロンプトを工夫することにより、領域矩形座標 (Bounding Box) を出力することも可能だが、座標位置はあまり正確ではなく、完全に出鱈目になるケースもある。

このハルシネーション時の挙動や座標位置の問題は後述するように、Qwen3-VL とも共通するものであり、視覚エンコーダーで利用している Vision Transformer（ViT）の仕組みに起因するものと考えられる。

4 Qwen3-VL

4.1 ROCm 環境用の例

Radeon 8060S (gfx1151) 用の ROCm 7.10.0 環境で実行可能なコードの例を示す：

```
from transformers import Qwen3VLForConditionalGeneration, AutoProcessor

model_path = "Qwen/Qwen3-VL-8B-Instruct"
processor = AutoProcessor.from_pretrained(model_path)
```

```
model = Qwen3VLForConditionalGeneration.from_pretrained(
    model_path,
    dtype="auto",
    device_map="auto",
    attn_implementation="flash_attention_2",
)


import torch
from transformers import (
    LogitsProcessor,
    LogitsProcessorList,
)
from qwen_vl_utils import process_vision_info


class PresencePenaltyProcessor(LogitsProcessor):
    """
    Apply a presence penalty: discourage generating tokens that have already appeared
    in the generated sequence (not frequency-based, but presence-based).
    This mimics OpenAI-style presence_penalty in a simple way by subtracting a fixed
    penalty from logits of any token present at least once in the generated tokens.
    """
    def __init__(self, presence_penalty: float):
        super().__init__()
        if presence_penalty < 0:
            raise ValueError("presence_penalty must be >= 0.")
        self.presence_penalty = presence_penalty

    def __call__(self, input_ids: torch.LongTensor,
                 scores: torch.FloatTensor) -> torch.FloatTensor:
        # input_ids shape: (batch, cur_len)
        # scores shape: (batch, vocab_size)
        batch_size = input_ids.shape[0]
        for b in range(batch_size):
            seen = set(input_ids[b].tolist())
            if len(seen) == 0:
                continue
            # Subtract penalty from logits of seen tokens
            # Note: scores[b] is (vocab_size,)
            # Efficient masking
            indices = torch.tensor(list(seen), device=scores.device, dtype=torch.long)
            # Clamp indices to valid range just in case
            indices = indices[(indices >= 0) & (indices < scores.shape[-1])]
```

```
            if indices.numel() > 0:
                scores[b, indices] -= self.presence_penalty
        return scores


def inference(
        messages,
        max_new_tokens=16384,
        do_sample=True,
        top_p=0.8,
        top_k=20,
        temperature=0.7,
        repetition_penalty=1.0,
        presence_penalty=1.5
    ):
    """
    Generates a response from the Qwen3-VL model based on the provided messages
    and generation options.

    Args:
        messages (list): A list of message dictionaries in the expected format.
        max_new_tokens (int): The maximum number of new tokens to generate.
        do_sample (bool): Whether to use sampling.
        top_p (float): The cumulative probability for top-p sampling.
        top_k (int): The number of highest probability vocabulary tokens to keep
                     for top-k sampling.
        temperature (float): The temperature for sampling.
        repetition_penalty (float): The penalty for repeating tokens.
        presence_penalty (float): The penalty for tokens that have already appeared.

    Returns:
        str: The generated text response.
    """
    text = processor.apply_chat_template(
        messages,
        tokenize=False,
        add_generation_prompt=True
    )
    images, videos, video_kwargs = process_vision_info(
        messages,
        image_patch_size=16,
        return_video_kwargs=True,
        return_video_metadata=True
    )
```

```
if videos is not None:
    videos, video_metadatas = zip(*videos)
    videos, video_metadatas = list(videos), list(video_metadatas)
else:
    video_metadatas = None

inputs = processor(
    text=text,
    images=images,
    videos=videos,
    video_metadata=video_metadatas,
    return_tensors="pt",
    do_resize=False,
    **video_kwargs
)
inputs = inputs.to(model.device)

logits_processors = LogitsProcessorList()
if presence_penalty and presence_penalty > 0:
    logits_processors.append(PresencePenaltyProcessor(presence_penalty))

generated_ids = model.generate(
    **inputs,
    max_new_tokens=max_new_tokens,
    do_sample=do_sample,
    top_p=top_p,
    top_k=top_k,
    temperature=temperature,
    repetition_penalty=repetition_penalty,
    logits_processor=logits_processors,
)

generated_ids_trimmed = [
    out_ids[len(in_ids) :]
    for in_ids, out_ids in zip(inputs.input_ids, generated_ids)
]

output_text = processor.batch_decode(
    generated_ids_trimmed,
    skip_special_tokens=True,
    clean_up_tokenization_spaces=False
)
return output_text[0]
```

```
messages = [
    {
        "role": "user",
        "content": [
            {
                "type": "image",
                "image":
    "https://codh.rois.ac.jp/pmjt/iiif/200014211/200014211_00005.tif/full/pct:25/0/default.jpg",
            },
            {
                "type": "text",
                "text": "縦書きの漢文として読んで、1行ごとに座標付きで出力して。"},
        ],
    }
]

response = inference(messages)
print(response)
```

4.2 MLX-VLM

Hugging Face の MLX Community [5] で公開されている MLX 用に変換されたモデルを使うことにより、Apple シリコンを搭載した Mac 上において、MLX-VLM を用いて実行することが可能である。例えば、Qwen3-VL-8B-Instruct の場合、`mlx-community/Qwen3-VL-8B-Instruct-8bit` を使えば良い。

DeepSeek-OCR の場合と同様に、コマンドライン・インターフェースを用いて、

```
(py3.13-mlx-vlm) % uv run python -m mlx_vlm generate --model mlx-community/Qwen3-VL-
8B-Instruct-8bit \
 --max-tokens 100000 --temperature 0.0 --prompt "<image> Locate every text line.
Report bbox coordinates in JSON format. For example:
[
  {
    "text": "...",
    "bbox": [x1, y1, x2, y2]
  }
]" --image /opt/images/kotenseki/items/200014683/image/00153_b_pct30.png
```

のように実行することができ、

```
Fetching 15 files: 100%|
```

5) https://huggingface.co/mlx-community

```
|| 15/15 [00:00<00:00, 200364.84it/s]
==========
Files: ['/opt/images/kotenseki/items/200014683/image/00153_b_pct30.png']

Prompt: <|im_start|>user
<|vision_start|><|image_pad|><|vision_end|><image> Locate every text line.
Report bbox coordinates in JSON format. For example:
[
  {
    text: ...,
    bbox: [x1, y1, x2, y2]
  }
]<|im_end|>
<|im_start|>assistant

```json
[
 {
  "text": "康熙字典",
  "bbox": [777, 270, 847, 480]
 },
 {
  "text": "子集上",
  "bbox": [715, 330, 785, 480]
 },
 {
  "text": "一部",
  "bbox": [645, 370, 715, 480]
 },
 {
  "text": "一古文弋",
  "bbox": [575, 270, 645, 420]
 },
 {
  "text": "（唐韻）韻會於悉切集韻正韻益悉切古漪入聲",
  "bbox": [575, 420, 645, 880]
 },
 {
  "text": "廣韻數之始也物之極也易繫辭天一地二老子道德經道生",
  "bbox": [515, 270, 575, 880]
 },
 {
  "text": "一一生二又廣韻同也禮樂記禮樂刑政其極一也史記儒",
  "bbox": [475, 270, 515, 880]
```
```

```
 },
 {
  "text": "林傳韓生推詩之意而為內外傳數萬言其語頗與齊魯間殊",
  "bbox": [435, 270, 475, 880]
 },
 {
  "text": "然其歸一也又少也顏延之庭誥文選書務一不尚煩密何",
  "bbox": [395, 270, 435, 880]
 },
 {
  "text": "承天答顏永嘉書竊願吾子舍乘而違一也又增韻純也易",
  "bbox": [355, 270, 395, 880]
 },
 {
  "text": "繫辭天下之動貞夫一老子道德經天得一以清地得一以寧",
  "bbox": [315, 270, 355, 880]
 },
 {
  "text": "神得一以靈谷得一以盈萬物得一以生侯王得一以爲天下",
  "bbox": [275, 270, 315, 880]
 },
 {
  "text": "正又均也唐書薛平傳兵雖完礪徭賦均一又誠也中庸",
  "bbox": [235, 270, 275, 880]
 },
 {
  "text": "康熙字典",
  "bbox": [255, 270, 275, 370]
 },
 {
  "text": "子集上",
  "bbox": [255, 410, 275, 510]
 },
 {
  "text": "一部",
  "bbox": [255, 510, 275, 570]
 }
]
```
==========
Prompt: 967 tokens, 279.197 tokens-per-sec
Generation: 800 tokens, 28.884 tokens-per-sec
Peak memory: 11.757 GB

のような結果を得ることができる。

また、下記のような Python プログラム経由で呼び出すこともできる：

```
import mlx.core as mx
from mlx_vlm import load, generate
from mlx_vlm.prompt_utils import apply_chat_template
from mlx_vlm.utils import load_config
import sys

# Load the model
model_path = "mlx-community/Qwen3-VL-8B-Instruct-8bit"
model, processor = load(model_path)
config = load_config(model_path)
max_tokens = 131072

# Prepare input
# image = ["https://kokusho.nijl.ac.jp/api/iiif/200017124/v4/NIIP/YU3-0212/YU3-0212-
00001.tif/full/pct:25/0/default.jpg"]
# image = [Image.open("...")] can also be used with PIL.Image.Image objects
image = [ sys.argv[1] ]
#prompt = "Describe this image."
#prompt = '<image> Output each line in JSON format: text, coordinates, line type and comment.'
prompt = '''<image> Locate every text line.
Report bbox coordinates of each line in JSON format.
For example:
[
  {"text": "...", "bbox": [x1, y1, x2, y2], "size": "...", "type": "...", "comment": "..."}
]
'''

# Apply chat template
formatted_prompt = apply_chat_template(
    processor, config, prompt, num_images=len(image)
)

# Generate output
output = generate(model, processor, formatted_prompt, image, verbose=False)
print(output)
```

今回は座標情報付き Markdown 形式 [1] で出力するため、このプログラムをベースに下記のような関数 `run_OCR_for_side` を定義した：

```
def run_OCR_for_side (BID, number, side, img_size, half_width, original_height, prompt,
                      MD_OUTPUT_PATH, OUTPUT_PATH):
    if side == 'a':
```

```
        x_ofs = half_width
    else:
        x_ofs = 0

    image_file = f"/opt/images/kotenseki/items/{BID}/image/{number}_{side}_pct{img_size}.png"
    subprocess.run(['convert',
                    f'/opt/images/kotenseki/items/{BID}/image/{number}.jpg',
                    '-crop', f'{half_width}x{original_height}+{x_ofs}+0',
                    '-resize', f'{img_size}%', image_file])

    print (image_file, prompt)

    # Prepare input
    image = [ image_file ]

    # Generate output
    response = generate(model, processor, formatted_prompt, image,
                        max_tokens = 4096, temperature=0.0,
                        verbose=False)
    print (response, type(response))
    response = response.text

    with open(f'{MD_OUTPUT_PATH}/{number}_{side}.md', 'w', encoding = 'utf-8') as md_destfile:
        for line_match
            in re.findall('\{"text": "(.*?)", "bbox": \[(\d+), (\d+), (\d+), (\d+)\],',
                          response):
            line_text, x1, y1, x2, y2 = line_match
            x1 = int (x1)
            y1 = int (y1)
            x2 = int (x2)
            y2 = int (y2)
            print (line_text, x1, y1, x2, y2)
            orx1 = round ( ( x1 * half_width      ) / 1000 + x_ofs )
            ory1 = round ( ( y1 * original_height ) / 1000 )
            orw  = round ( ( ( x2 - x1 ) * half_width)      / 1000 )
            orh  = round ( ( ( y2 - y1 ) * original_height) / 1000 )
            print (f'![.]({IIIF_img_url}/{orx1},{ory1},{orw},{orh}/pct:40/0/default.jpg)<br\
/>{line_text}<br/>',
                   file=md_destfile)

        with open(f'{OUTPUT_PATH}/{number}_{side}.json', 'w', encoding = 'utf-8')
             as destfile:
            destfile.write(response)
```

```
        with open(f'{OUTPUT_PATH}/{number}_{side}.prompt', 'w', encoding = 'utf-8')
             as prompt_file:
            prompt_file.write(prompt)

        subprocess.run("git pull", shell=True)
        subprocess.run(f"git add {OUTPUT_PATH}/{number}_{side}.json \
{OUTPUT_PATH}/{number}_{side}.prompt {MD_OUTPUT_PATH}/{number}_{side}.md",
                       shell=True)
        subprocess.run("git pull", shell=True)
        subprocess.run("git commit -am 'New files.'", shell=True)
        subprocess.run("git pull", shell=True)
        subprocess.run("git push", shell=True)
```

DeepSeek-OCR も同様なのであるが、Qwen3-VL でも見開きの左右の順番（頁の進行方向）を取り違える場合があり、この関数では見開き頁の左右のいずれかを指定し、その部分を切り取って処理するようにしている。それを指定するのが引数 `side` で、これが `'a'` の時は右半分、`'b'` の時は左半分を示す。

結果の座標位置は縦横それぞれ 1000 に正規化した値となっているため、これをオリジナルサイズに戻し、座標情報付き Markdown 形式で出力するようにしている。また、結果を `https://gitlab.nijl.ac.jp/Kotenseki/item` に設けた Git リポジトリに push するようにしている。

5 OCR 処理結果の Git リポジトリ化

DeepSeek-OCR と Qwen3-VL による漢籍画像の OCR 結果を

`https://gitlab.nijl.ac.jp/Kotenseki/item`

に置いている。

漢籍の体現系ないしは個別資料毎に 1 リポジトリとし、国書データベースに収録されている使用に関しては国書 BID をリポジトリ名とする。例えば、`https://gitlab.nijl.ac.jp/Kotenseki/item/200018997` は「日本書紀」`https://doi.org/10.20730/200018997` の OCR 結果を指す。

各リポジトリに、

- `DeepSeek-OCR-8bit/`
- `Qwen3-VL-8B/`
- `Qwen3-VL-30B-A3B/`

のように視覚言語モデルの名前を示すディレクトリを掘りその中に結果を格納する。

一般に、視覚言語モデルを利用した OCR の結果はプロンプトに左右されるため、Qwen3-VL に関しては、厳密にプロンプトを区別するためにプロンプトの文字列をファイルとして IPFS に格納した際に得られる CID をサブフォルダーとしその中に、

- `markdown_line_pct`*percent*
- `raw_pct`*percent*

というサブフォルダーを設け、その中に結果を格納している。ここで、*percent* は入力画像の縮小率を示す。例えば、30%に縮小した画像を用いた時の結果は `markdown_line_pct30/` と `raw_pct30/` に格納される。

DeepSeek-OCR も Qwen3-VL も視覚エンコーダーに Vision Transformer（ViT）を用いており、概ね縦横 1000 に区切った升目に分割するため、入力画像は縦横 1000 ピクセル程度の時が最も認識しやすくなると考えられるが、その一方で、あまりに荒くしすぎて文字が潰れてしまうとこれまた認識しづらくなり、ハルシネーションが発生しやすくなる。理想的には読み取りたい文字のある部分を切り出して 1000 ピクセル程度に縮小して渡すのが良いと考えられるが、今回は資料毎に幾つかの縮小率を試すことを想定しこのような仕様とした。

一方、DeepSeek-OCR の場合、プロンプトは

'<image>\n<grounding|>Convert the document to markdown.'|

に固定することとし、各リポジトリの `DeepSeek-OCR-8bit/` というディレクトリの直下に `markdown_line_pct`*percent*`/` というサブフォルダーを設けその中に結果を格納することにした。

6 認識結果

康熙字典 (`https://doi.org/10.20730/200014683`) の p.153 左 〜 p.159 と日本書紀 (`https://doi.org/10.20730/200018997`) の p.4 〜 p.6 に対する Qwen3-VL-8B と DeepSeek-OCR-8bit と NDL 古典籍 OCR v3 の適合率、再現率、F 値をそれぞれ表 1, 2 に示す。なお、NDL 古典籍 OCR v3 のデータは `https://gitlab.nijl.ac.jp/Kokusho/200014683_text/-/tree/main/kotenocr3bid/200014683` と `https://gitlab.nijl.ac.jp/Kokusho/200018997_text/-/tree/master/kotenocr3bid/200018997` のものを用いた。なお、Qwen3-VL-8B と DeepSeek-OCR-8bit にはそれぞれ 30% に縮小した画像を入力した。なお、NDL 古典籍 OCR v3 のデータはオリジナルサイズのものである。

特筆すべきは、Qwen3-VL-8B の適合率の高さで、これはハルシネーションがうまく抑制されていて、読めない部分をなるべくさっくり落とすような挙動をしていることを示し

モデル名	適合率	再現率	F 値
Qwen3-VL-8B	93.8	79.4	86.0
DeepSeek-OCR-8bit	9.1	52.3	15.5
NDL 古典籍 OCR v3	69.2	70.8	70.0

表 1　康熙字典の認識結果

モデル名	適合率	再現率	F 値
Qwen3-VL-8B	94.4	87.4	90.8
DeepSeek-OCR	1.5	57.5	2.9
NDL 古典籍 OCR v3	55.7	78.7	65.2

表 2　日本書紀の認識結果

ている。また、古典的な OCR はレイアウト解析を行なって切り出した文字に対する分類問題を解いているため、数万以上の漢字の中から適切な抽象文字を選ぶようなことは原理的に困難であるが、Qwen3-VL-8B は（うまく読めている場合には）文脈と原画像の双方を勘案して適切な符号位置を選ぶような挙動を示し、著者が作成した正解データと比較している際、著者が作成した『正解』データの方が誤っていて Qwen3-VL-8B の方が元テキストに照らし合わすと適切というケースがいくつかあって驚かされた。ただ、レイアウト解析が現代の横書き文書向きにチューンされているせいか、割注は苦手であり、古字書のような割注が多いものは失敗しやすい。

一方、DeepSeek-OCR の適合率が極端に悪いのは、理解できない文書に出会ったときに、その文書に現れた（DeepSeek-OCR が部分的に理解したと思った）部分を延々と繰り返し出力するようなハルシネーションが多発したせいである。これはおそらく視覚エンコーダーで用いている Vision Transformer (ViT) の特性に起因していると考えられる。ViT では升目に区切った画像の部分（画像パッチ）を、Transformer で自然言語を処理する際の単語やサブワードに相当するものとして処理しているが、文書理解がうまくできない場合、画像トークンを適当にくっつけたものとそれに対応する文字列（あるいは、その逆パターンや混合パターン）が生じるのではないかと考えられる。

また、今回は矩形座標の評価結果は示せなかったが、Qwen3-VL-8B, DeepSeek-OCR 双方とも、文字列がきちんと読めている場合でも矩形座標がずれていたり出鱈目なものが出力されるケースが少なくなかった。これもおそらく ViT における位置エンコーディングの仕組みに起因したものだと考えられる。

なお、今回はデータを示せなかったが、Qwen3-VL-30B-A3B の場合、8B ではうまく扱えなかった本文と割注の境界をうまく区切れることができることもある反面、DeepSeek-OCR と同様なハルシネーションが多発した。Qwen3-VL-30B-A3B は総パラ

メーター数では約 300 億と 8B の約 80 億に勝るものの、実行コストを抑えるために一度にアクティブになるパラメーター数は 3B（約 30 億）と 8B の 約 80 億よりも控えめになっていることが影響しているのかもしれない。

また、NDL 古典籍 OCR の適合率が悪いのは返り点やふりがな、送り仮名等が出力される場合があるからである。今回の実験では本文のみを扱いこれらは省いた結果、余計な文字を出したと判定されてしまった訳である。なお、もし、NDL 古典籍 OCR が返り点を Unicode の漢文ブロックの文字などを使って適切に出力できるならばこれらの情報を出すことはむしろ望ましいと言えるが、現状、特に区別なしに混合した文字列として出力するため、非常に使いづらいデータになってしまう。また、常にこうした情報が出力される訳ではなくないことも問題であり、こうしたことが認識性能以上に校正用データとしての使いづらさにつながっているといえよう。

7 おわりに

視覚言語モデルの Qwen3-VL と DeepSeek-OCR を利用して漢籍の版本画像の自動テキスト化を試みた。

DeepSeek-OCR は条件が良いと古典中国語をうまく理解するが、基本的に現代文書用といえ、漢籍を処理する場合、ハルシネーションが多発し、NDL 古典籍 OCR (ver.3) よりもトータルでは認識精度は悪い。

一方、Qwen3-VL-8B は割注の認識等に課題は残るものの、比較的良好に古典中国語の画像を認識する。ただし、矩形画像の出力品質は良くない。テキスト部分だけに限れば、現状、漢籍用 OCR として最も優れた選択肢の一つといえる。

なお、Mac を使う場合、NVIDIA 製の GPU が使えないために NDL 古典籍 OCR(ver.3) は使えず CPU だけで動作する NDL 古典籍 OCR-Lite を使う必要があったが、Qwen3-VL-8B 等の視覚言語モデルの場合、Apple シリコン用の機械学習用フレームワーク MLX およびこれを用いた視覚言語モデル用インターフェース MLX-VLM を用いて効率良く視覚言語モデルを用いた推論が行えるため、8B クラスのモデルであれば最低 16GB, 快適に使うには 32GB 以上のメモリーを搭載した MacBook Pro や Mac mini で高速に処理可能であり、GPU に対する搭載メモリーの量を考えれば比較的低コストであり、漢籍用 OCR として現実的な選択肢であると考えられる。

参考文献

[1]守岡 知彦. “Markdown を用いた古典籍 OCR テキストの簡易マークアップの試み”. In: **情処研報** 2025-CH-139.8 (July 2025), pp. 1–5.