Vision Language Models

for Document Understanding

A Practical Guide

Eivind Kjosbakken

November 23, 2025

Contents

Preface iii

1 Introduction to Vision Language Models 1
- 1.1 What are Vision Language Models? . . . 1
- 1.2 Why Traditional OCR Falls Short . . . 1
- 1.3 Key Application Areas . . . 2

2 Applying VLMs to Long Documents 3
- 2.1 The Long Document Challenge . . . 3
- 2.2 OCR Using VLMs . . . 3
  - 2.2.1 Markdown Extraction . . . 3
  - 2.2.2 Explaining Visual Information . . . 4
  - 2.2.3 Adding Missing Information . . . 4
- 2.3 Direct VLM Processing . . . 5
  - 2.3.1 Model Selection . . . 5
  - 2.3.2 Processing Power Considerations . . . 5
  - 2.3.3 Hierarchical Processing . . . 6
- 2.4 Cost Considerations . . . 6

3 Extracting Metadata from Complex Documents 7
- 3.1 Why Extract Metadata . . . 7
- 3.2 Three Approaches to Metadata Extraction . . . 7
  - 3.2.1 Approach 1: Regex . . . 7
  - 3.2.2 Approach 2: OCR + LLM . . . 8
  - 3.2.3 Approach 3: Vision LLMs . . . 9
- 3.3 Challenges in Metadata Extraction . . . 9
  - 3.3.1 Choosing the Right Approach . . . 9
  - 3.3.2 Handling Handwritten Text . . . 9
  - 3.3.3 Long Document Considerations . . . 9

4 Optimizing Prompts for Performance 11
- 4.1 Why Optimize Your Prompts . . . 11
- 4.2 Technique 1: Always Keep Static Content Early . . . 11
  - 4.2.1 Implementation . . . 11
  - 4.2.2 Document Processing . . . 12
- 4.3 Technique 2: Question at the End . . . 12
- 4.4 Technique 3: Using Prompt Optimizers . . . 13
  - 4.4.1 Simple Approach . . . 13

4.4.2 Better Approach . . . 13
4.5 Technique 4: Benchmark Different LLMs . . . 13

5 Using Frontier VLMs: Qwen-VL 14
5.1 Introduction to Qwen-VL . . . 14
5.2 Setup and Installation . . . 14
5.2.1 Loading the Model . . . 14
5.3 OCR with Qwen 3 VL . . . 15
5.3.1 Running OCR . . . 15
5.4 Information Extraction . . . 16
5.5 Current Limitations . . . 16

6 Fine-Tuning VLMs 18
6.1 When to Fine-Tune . . . 18
6.2 The Importance of Data Inspection . . . 18
6.3 Efficient Annotation Process . . . 19
6.4 Supervised Fine-Tuning (SFT) Details . . . 20
6.4.1 Using Unsloth for Training . . . 20
6.4.2 Critical Considerations . . . 20
6.4.3 Label Correctness is Critical . . . 20
6.4.4 Data Balancing . . . 20
6.4.5 Layer Selection . . . 20
6.4.6 Hyperparameter Search . . . 21
6.5 Results . . . 21

7 Processing Documents at Scale 24
7.1 Cost Implications . . . 24
7.2 Optimization Strategies . . . 24
7.2.1 Maximize Prompt Caching . . . 24
7.2.2 Right-Size Your Approach . . . 24
7.2.3 Resolution Optimization . . . 25
7.2.4 Hierarchical Processing . . . 25
7.3 Infrastructure Considerations . . . 25
7.3.1 Essential Components . . . 25
7.3.2 Open Source vs API . . . 25
7.4 Best Practices . . . 26

8 Conclusion and Best Practices 27
8.1 Key Takeaways . . . 27
8.2 Recommended Workflow . . . 27
8.3 The Future of VLMs . . . 28
8.4 Final Thoughts . . . 28

A Resources 29
A.1 Referenced Articles . . . 29

Preface

Vision Language Models (VLMs) are transforming how we process and understand documents. Unlike traditional approaches that rely on OCR followed by text processing, VLMs can directly interpret both visual and textual information, making them ideal for complex document understanding tasks.

This book distills my own insights from processing millions of documents and provides practical guidance from my Towards Data Science articles listed below:

- Understanding when and why VLMs are necessary (Using VLMs to Process Millions of Documents)
- Applying VLMs to documents of varying lengths and complexity (Applying VLMs to Long Documents)
- Optimizing performance for cost, latency, and accuracy (4 Optimization Techniques)
- Extracting metadata consistently from complex documents (Extracting Metadata)
- Fine-tuning VLMs for specialized tasks (Fine-Tuning VLMs)
- Using frontier models like Qwen-VL (Using Qwen-VL)

The content is based on real-world experience processing millions of documents and represents practical and tested approaches to VLM deployment.

Chapter 1

Introduction to Vision Language Models

1.1 What are Vision Language Models?

Vision Language Models (VLMs) are powerful machine learning models that can process both visual and textual information simultaneously. Unlike traditional Large Language Models (LLMs) that only accept text, VLMs can input images and text together, then respond with text output.

Key Point

VLMs are required when the meaning of text depends on its visual position. This makes them essential for document understanding tasks where layout, formatting, and visual elements carry semantic information.

1.2 Why Traditional OCR Falls Short

The alternative to VLMs is using Optical Character Recognition (OCR) to extract text, then feeding that text into an LLM. However, this approach has significant limitations:

- **Imperfect text extraction:** OCR engines aren't perfect and may struggle with small text, skewed images, or vertical text
- **Loss of visual position:** You lose information about where text is positioned relative to other elements
- **Missing non-text information:** Traditional OCR cannot interpret symbols, drawings, or visual structures

Consider the checkbox example shown in Figure 1.1. The image shows three document options with checkboxes, where only one checkbox is marked with a filled in circle.

If you apply traditional OCR like Tesseract to Figure 1.1, you would simply see something like the text below, where you only extract the names of each document, and no other content:

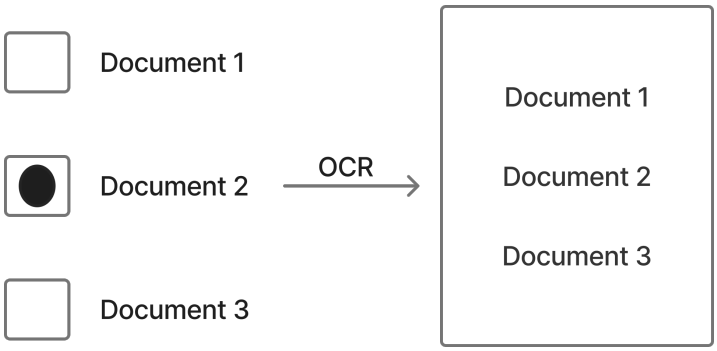


Figure 1.1: A document form with three checkbox options. Traditional OCR extracts the text labels ("Document 1", "Document 2", "Document 3") and completely misses the filled-in circle, because it's not a character. In this example, the checkboxes for Document 1 and Document 3 are unmarked, while Document 2 is checked off.

```
Document 1
Document 2
Document 3
```

Notice the problem? It's impossible to know which document(s) are checked off! The OCR output shows the text, but no other information, so the visual information about the checkboxes is completely lost. An LLM processing this text cannot determine which specific documents are marked.

With a VLM, this task becomes trivial - the model can directly see the image and immediately identify that Document 2 is checked off, while the other two documents are not.

1.3 Key Application Areas

VLMs excel at several document understanding tasks:

1. **Question Answering:** Feed the VLM a document image and a question, receive an answer
2. **Classification:** Categorize documents into predetermined classes
3. **Information Extraction:** Extract specific data points in structured formats like JSON
4. **OCR:** Perform advanced text extraction with formatting and context

Chapter 2

Applying VLMs to Long Documents

2.1 The Long Document Challenge

Processing lengthy documents (over 100 pages) or very dense documents with detailed drawings presents significant challenges. VLMs need many tokens to represent visual information - typically around 10x more input tokens than using text-only LLMs.

Warning

VLMs are limited by context windows just like traditional LLMs. Processing hundreds of pages requires careful strategy.

2.2 OCR Using VLMs

One powerful approach is using VLMs to perform advanced OCR, extracting text that can then be processed by traditional LLMs. This provides the best of both worlds:

2.2.1 Markdown Extraction

VLMs can extract formatted Markdown instead of plain text:

- Provide headers and subheaders for document structure
- Represent tables accurately
- Preserve formatting like bold and italic text

Key Point

LLMs perform significantly better on formatted text like Markdown than on plain text extracted using traditional OCR.

2.2.2 Explaining Visual Information

VLMs can describe non-text elements. For example, given a document with an embedded image:

Original Document

```
This is the introduction text of the document
<image showing the Eiffel tower>
This is the conclusion of the document
```

Traditional OCR would output:

Traditional OCR Output

```
This is the introduction text of the document
This is the conclusion of the document
```

A VLM can provide:

VLM Output

```
This is the introduction text of the document
<image>
This image depicts the Eiffel tower during the day
</image>
This is the conclusion of the document
```

2.2.3 Adding Missing Information

VLMs can identify blank fields. Consider the form shown in Figure 2.1.



Figure 2.1: A form with three fields where the Date field is blank. VLMs can detect and explicitly mark empty fields, while traditional OCR simply omits them from the output, making it impossible to distinguish between a blank field and a missing field.

Traditional OCR outputs:

Traditional OCR Output

```
Address Road 1
Date
Company Google
```

A VLM can be instructed to output:

VLM Output

```
Address Road 1
Date <empty>
Company Google
```

This explicitly indicates that the date field exists but is blank, providing crucial information for downstream processing.

2.3 Direct VLM Processing

While VLM-based OCR is powerful, directly processing documents with VLMs often yields better results. As the saying goes, "an image is worth a thousand words" - visual information is richer than any text description.

2.3.1 Model Selection

According to the HuggingFace VLM leaderboard:

- **Closed source:** Gemini 2.5 Pro and GPT-4o offer excellent performance through APIs
- **Open source:** SenseNova 6.5 and Qwen 3 VL provide strong performance with more control

2.3.2 Processing Power Considerations

When running open-source models, consider:

1. **Hardware requirements:** Typically need at least an A100 GPU (costs $1.5-2/hour)
2. **Latency tolerance:** Live chat requires fast processing; batch processing can tolerate delays
3. **Image resolution:** Higher resolution (2048×2048+) needed for text reading but increases processing time
4. **Number of pages:** More pages increase processing time linearly

2.3.3 Hierarchical Processing

To reduce processing requirements, start simple and only advance to heavier processing if needed:

1. Start with only the first 10 pages at lower resolution
2. If the required information isn't found, process more pages
3. If text is unreadable, increase resolution

This dramatically reduces average processing costs since most tasks can be solved with minimal processing.

2.4 Cost Considerations

Cost is critical when using VLMs at scale. Key insights:

- VLMs typically require 10x more input tokens than text-based LLMs
- Input tokens are usually the main cost driver for long documents
- Maximizing cached token usage is essential (covered in Chapter 4)

For OCR tasks specifically, note that output tokens can be high since the VLM outputs all text in the images.

Chapter 3

Extracting Metadata from Complex Documents

3.1 Why Extract Metadata

Documents contain vast amounts of important information buried in their contents. Pre-extracting metadata simplifies downstream tasks like:

- Filtering documents by type, date, address, etc.
- Improving RAG (Retrieval Augmented Generation) quality
- Reducing inference costs by providing pre-extracted data

Key Point

While RAG can find information, pre-extracted metadata is more reliable, faster, and cheaper for structured queries.

3.2 Three Approaches to Metadata Extraction

3.2.1 Approach 1: Regex

The simplest approach works when you know the exact format beforehand. For example, if dates always appear as `dd.mm.yyyy` right after "Date: ", regex is ideal.

Unfortunately, most real-world scenarios involve inconsistent documents with challenges like:

- Data appears in different locations
- OCR quality varies, missing characters
- Multiple date formats (mm.dd.yyyy, "22nd of October", "December 22", etc.)

3.2.2 Approach 2: OCR + LLM

A powerful middle-ground approach:

1. Apply OCR to extract document text
2. Prompt an LLM to extract specific metadata

This works incredibly well because LLMs can:

- Understand context (which dates are relevant vs irrelevant)
- Parse different date formats automatically
- Handle both European (dd.mm.yyyy) and American (mm.dd.yyyy) standards

Figure 3.1 illustrates this approach:

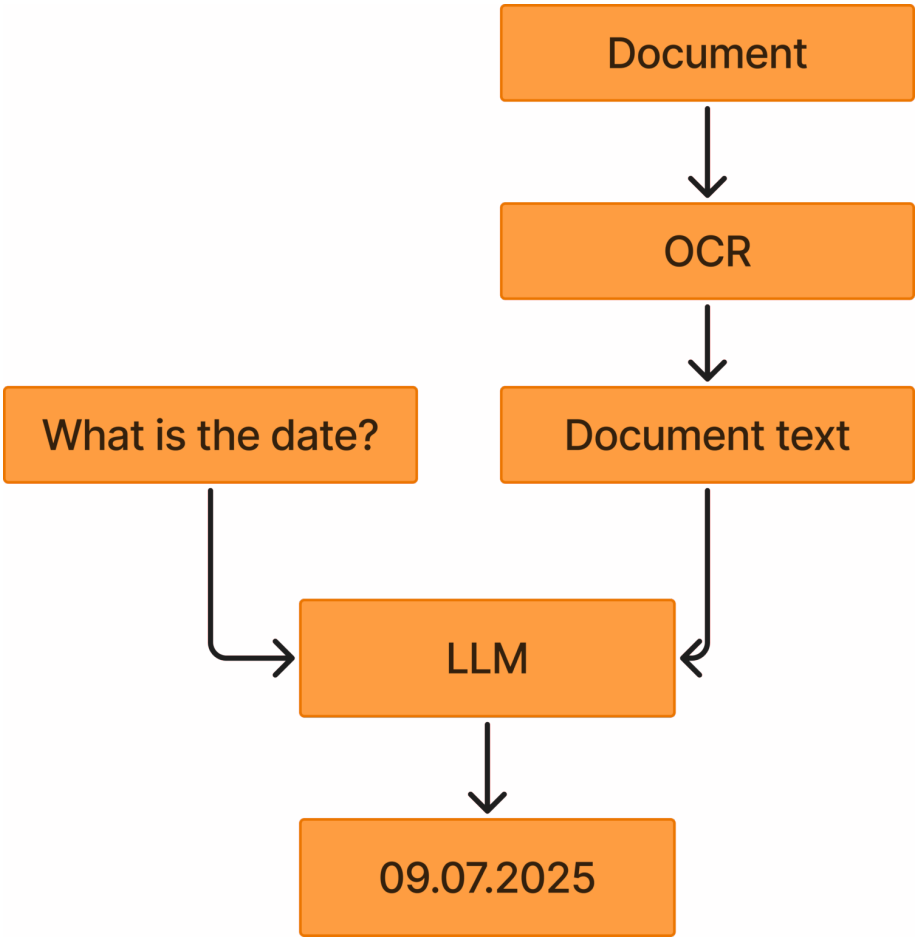


Figure 3.1: The OCR + LLM metadata extraction pipeline. A document is first processed through OCR to extract text, which is then combined with a prompt and fed to an LLM to extract structured metadata. This approach balances cost and capability, leveraging the LLM's contextual understanding while avoiding the higher token costs of processing images directly.

3.2.3 Approach 3: Vision LLMs

The most powerful but expensive approach. Required when metadata depends on visual information that OCR cannot capture.

Example use case: Extracting which checkboxes are filled

As shown in Figure 3.2, OCR will extract the text "Document 1, Document 2, Document 3" but completely miss which checkboxes are filled. Vision LLMs can easily solve this task by directly interpreting the visual information.



Figure 3.2: A form showing three document options with checkboxes. Documents 1 and 3 have their checkboxes marked, while Document 2 remains unchecked. Traditional OCR can extract the document labels but cannot determine which checkboxes are selected, making Vision LLMs essential for this type of metadata extraction.

Warning

Vision LLMs typically cost significantly more than pure text LLMs due to high-resolution image requirements and resulting token counts.

3.3 Challenges in Metadata Extraction

3.3.1 Choosing the Right Approach

Ideally, use Vision LLMs for everything, but cost makes this impractical. Guidelines:

- **Use OCR + LLM** for: Dates, names, addresses, standard text fields
- **Use Vision LLMs** for: Checkboxes, signatures, visual layouts, handwritten text

3.3.2 Handling Handwritten Text

Traditional OCR struggles with handwriting. If documents contain handwritten content, Vision LLMs are strongly recommended as they handle handwriting significantly better.

Note that many documents contain both printed and handwritten text, requiring hybrid approaches.

3.3.3 Long Document Considerations

For extremely long documents, balance cost vs completeness:

- Most metadata (like dates) appears early in documents
- Processing only first 100 pages may be acceptable

- Rare cases where data is on page 101+ will be missed but costs are dramatically reduced

The key is understanding your specific document corpus and making informed trade-offs.

Chapter 4

Optimizing Prompts for Performance

4.1 Why Optimize Your Prompts

Most prompts work adequately but leave significant potential on the table. Using specific optimization techniques, you can:

- Improve response quality
- Reduce costs dramatically
- Decrease latency

Key Point

Just because a prompt works doesn't mean it's performing optimally. Small optimizations yield great improvements with minimal effort.

4.2 Technique 1: Always Keep Static Content Early

All major LLM providers (Anthropic, Google, OpenAI) implement prompt caching. Cached tokens cost approximately 10% of normal input tokens.

Key Point

Cached tokens are tokens already processed in a previous API request that can be processed cheaper and faster than normal tokens.

If you send the same prompt twice, the second request's input tokens cost only 1/10th of the first!

4.2.1 Implementation

Structure prompts with static content first, variable content last:

```
prompt = f"""
{long_static_system_prompt}
{user_question}
"""
```

Example:

```
prompt = f"""
You are a document expert...
You should always reply in this format...
If a user asks about... you should answer...
{user_question}
"""
```

4.2.2 Document Processing

If processing different documents each time:

```
prompt = f"""
{static_system_prompt}
{variable_prompt_instruction_1}
{document_content}
{variable_prompt_instruction_2}
{user_question}
"""
```

If processing the *same* documents multiple times:

```
prompt = f"""
{static_system_prompt}
{document_content}
{variable_prompt_instruction_1}
{variable_prompt_instruction_2}
{user_question}
"""
```

Warning

Cached tokens typically activate only when the first 1024 tokens are identical. Placing ANY variable content before this threshold breaks caching completely!

DO NOT do this:

```
prompt = f"""
{variable_content}  # This breaks ALL caching!
{static_system_prompt}
{document_content}
"""
```

4.3 Technique 2: Question at the End

Always place the user question at the very end of your prompt. Ideally:

```
system_prompt = "<general instructions>"
user_prompt = f"{user_question}"
```

According to Anthropic's prompt engineering documentation, placing the question at the end can improve performance by up to 30%, especially for long contexts. This makes it clearer to the model what task it's trying to achieve.

4.4 Technique 3: Using Prompt Optimizers

Human-written prompts tend to be messy, inconsistent, and redundant. Use prompt optimizers to improve them.

4.4.1 Simple Approach

Prompt an LLM: "Improve this prompt: {prompt}"

4.4.2 Better Approach

Use specialized prompt optimizers in OpenAI's or Anthropic's consoles. These are LLMs specifically trained to optimize prompts.

For best results, include:

- Details about the task you're trying to achieve
- Examples of successful inputs and outputs
- Examples of failed cases with their inputs and outputs

Key Point

Spending 10-15 minutes with a prompt optimizer can yield dramatically better prompts with minimal effort - one of the highest ROI activities in LLM development.

4.5 Technique 4: Benchmark Different LLMs

Different LLMs excel at different tasks. Set up access to major providers:

- Google Gemini
- OpenAI GPT
- Anthropic Claude

Switching providers takes minutes if you have credentials set up. Create a specific benchmark for your task and test which model performs best.

Also consider testing open-source models for better cost/control, though they require more setup effort.

Warning

Regularly check model performance. Providers occasionally upgrade models without new version numbers, potentially changing behavior.

Chapter 5

Using Frontier VLMs: Qwen-VL

5.1 Introduction to Qwen-VL

Qwen 3 VL represents the latest generation of open-source vision language models. Released with models ranging from 4B to 235B parameters, Qwen 3 VL offers powerful capabilities for document understanding tasks.

This chapter provides practical implementation examples using Qwen 3 VL for OCR and information extraction.

5.2 Setup and Installation

Required dependencies:

```
pip install torch accelerate pillow torchvision
pip install git+https://github.com/huggingface/transformers
```

Note: Install Transformers from GitHub source as Qwen 3 VL requires the latest version.

5.2.1 Loading the Model

```
from transformers import (
    Qwen3VLForConditionalGeneration,
    AutoProcessor
)
from PIL import Image
import time

model = Qwen3VLForConditionalGeneration.from_pretrained(
    "Qwen/Qwen3-VL-4B-Instruct",
    dtype="auto",
    device_map="auto"
)

processor = AutoProcessor.from_pretrained(
    "Qwen/Qwen3-VL-4B-Instruct"
)
```

5.3 OCR with Qwen 3 VL

VLMs are specifically trained on OCR data during pre-training. To perform OCR:

```
def inference(system_prompt: str, user_prompt: str,
              max_new_tokens: int, image_paths: list[str]):
    messages = [
        {"role": "system",
         "content": [{"type": "text", "text": system_prompt}]
        }
    ]

    user_content = []
    for image_path in image_paths:
        user_content.append({
            "type": "image",
            "min_pixels": 512*32*32,
            "max_pixels": 2048*28*28
        })

    user_content.append({"type": "text", "text": user_prompt})
    messages.append({"role": "user", "content": user_content})

    inputs = processor.apply_chat_template(
        messages,
        tokenize=True,
        add_generation_prompt=True,
        return_dict=True,
        return_tensors="pt"
    )
    inputs = inputs.to(model.device)

    generated_ids = model.generate(
        **inputs,
        max_new_tokens=max_new_tokens
    )

    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]
```

5.3.1 Running OCR

```
system_prompt = "You are a helpful assistant."
user_prompt = "Read all the text in the image."
max_new_tokens = 1024
```

```
image_paths = ["document.jpg"]

output = inference(
    system_prompt,
    user_prompt,
    max_new_tokens,
    image_paths
)
print(output)
```

5.4 Information Extraction

Extract structured metadata by prompting for JSON output:

```
user_prompt = """
Extract the following information from the image
and reply in JSON format:
{
    "date": "The date in YYYY-MM-DD format",
    "address": "The address mentioned",
    "gnr": "The street number (Gnr)",
    "scale": "The scale (malestokk)"
}
If you cannot find information, reply with None.
The return object should be valid JSON only.
"""

max_new_tokens = 1024
image_paths = ["site-plan.jpg"]

output = inference(
    system_prompt,
    user_prompt,
    max_new_tokens,
    image_paths
)
print(output)
```

Example output:

```
{
    "date": "2014-01-23",
    "address": "Camilla Colletts vei 15",
    "gnr": "15",
    "scale": "1:500"
}
```

Key Point

Qwen 3 VL can make intelligent inferences - in this example, it correctly extracted the street number from the address even though the Gnr field was blank in the original document.

5.5 Current Limitations

While powerful, VLMs have some downsides:

- **Missing text:** VLMs occasionally miss sections during OCR, which is problematic for critical applications
- **Slow inference:** Processing high-resolution images (2048×2048+) requires significant compute
- **Memory requirements:** Even the 4B model can struggle with very large or multiple high-resolution images

Warning

For production OCR at scale, validate VLM output against traditional OCR to catch missed sections.

Chapter 6

Fine-Tuning VLMs

6.1 When to Fine-Tune

Note on Model Versions

The fine-tuning experiments in this chapter were conducted using **Qwen 2.5-VL**, which was the state-of-the-art model at the time of writing. While Qwen 3 VL (introduced in the previous chapter) is the current frontier model, the principles and methodologies for fine-tuning remain identical.

Fine-tuning is beneficial when:

- You have domain-specific documents with unique characteristics
- Pre-trained models underperform on your specific task
- You need specialized extraction patterns (e.g., handwritten text)
- You're processing large volumes where even small accuracy improvements matter

Key Point

Manual inspection of data has probably the highest value-to-prestige ratio of any activity in machine learning. - Greg Brockman, President of OpenAI

6.2 The Importance of Data Inspection

Before fine-tuning, spend considerable time manually inspecting your dataset. This reveals:

- Common patterns and edge cases
- Visual characteristics (e.g., how specific digits are written)
- Data challenges the model will face
- Insights for creating better system prompts



Figure 6.1: Example images from the handwritten digits dataset. The cells vary in width from 81 to 93 pixels and in height from 48 to 57 pixels, making them very small images to process. These examples showcase the variety and challenges in the dataset, including faint text, similar-looking digits (1s vs 7s), dots in the background, and cell borders that can be misinterpreted as characters. All images shown are readable from a human perspective but present challenges for automated extraction.

Figure 6.1 shows example images from the handwritten digits dataset used for fine-tuning:

Example findings from inspecting handwritten digits:

1. The "1"s look similar to "7"s
2. Some text is faintly written
3. All images are readable from a human perspective
4. Cell borders can be misinterpreted as characters

6.3 Efficient Annotation Process

For tasks where the base model is already decent (90%+ accuracy), use this three-step cycle:

1. **Predict:** Use the base model to predict on a few hundred samples
2. **Review & Correct:** Manually check predictions and fix errors
3. **Retrain:** Fine-tune the model on corrected labels

Repeat until model performance converges.

Figure 6.2 illustrates this iterative process:

This approach is much faster than manually labeling from scratch since you only correct mistakes rather than annotating everything.

6.4 Supervised Fine-Tuning (SFT) Details

6.4.1 Using Unsloth for Training

Unsloth provides efficient VLM fine-tuning. For Qwen 2.5 VL:

```
from unsloth import FastVisionModel

model, tokenizer = FastVisionModel.from_pretrained(
    "unsloth/Qwen2.5-VL-7B-Instruct",
    load_in_4bit=False,
    use_gradient_checkpointing="unsloth",
)
```

6.4.2 Critical Considerations

When fine-tuning on tasks where the base model already performs well:

1. **Low learning rate:** Only slightly update weights
2. **Low LoRA rank:** Update only a small set of weights
3. **Perfect labels:** Model is extremely sensitive to errors
4. **Balance dataset:** Avoid over-representing common cases
5. **Tune all layers:** ViT, VL adapter, and decoder

6.4.3 Label Correctness is Critical

Warning

Just a few labeling errors can significantly hurt model performance. With 1000 labels where 99.5% were correct (only 5 errors!), the model actually performed worse after fine-tuning.

The model blindly trusts your labels. If you label a parenthesis image as a bracket, the model learns incorrect patterns that degrade performance on correct examples.

Figure 6.3 shows an example of this type of labeling error (from Fine-Tuning VLMs for Document Understanding):

6.4.4 Data Balancing

If 70% of your data consists of blank images and the model already handles these well, limit blank images to 30% of training data to focus learning on challenging cases.

6.4.5 Layer Selection

VLM architecture consists of:

- **ViT (Vision Transformer):** Extracts visual tokens from images

- **VL Adapter:** Aligns image tokens with text token embedding space
- **Decoder:** Generates output text from combined tokens

Ideally, tune all layers. For compute-constrained scenarios with vision-heavy tasks (e.g., architectural drawing classification), you may tune only ViT and VL adapter.

6.4.6 Hyperparameter Search

If training is fast enough (10-20 minutes), run overnight hyperparameter searches. For longer training runs, rely on intuition and incremental experimentation.

6.5 Results

Example results from fine-tuning Qwen 2.5 VL 7B on handwritten digits across four test sets:

Model	Test Set 1	Test Set 2	Test Set 3
EasyOCR	72%	68%	71%
Qwen Base	93%	97%	99%
Qwen Fine-tuned	**99%**	**99%**	**100%**

Table 6.1: Performance comparison across models

Fine-tuning clearly improved performance, achieving near-perfect accuracy on handwritten text extraction.

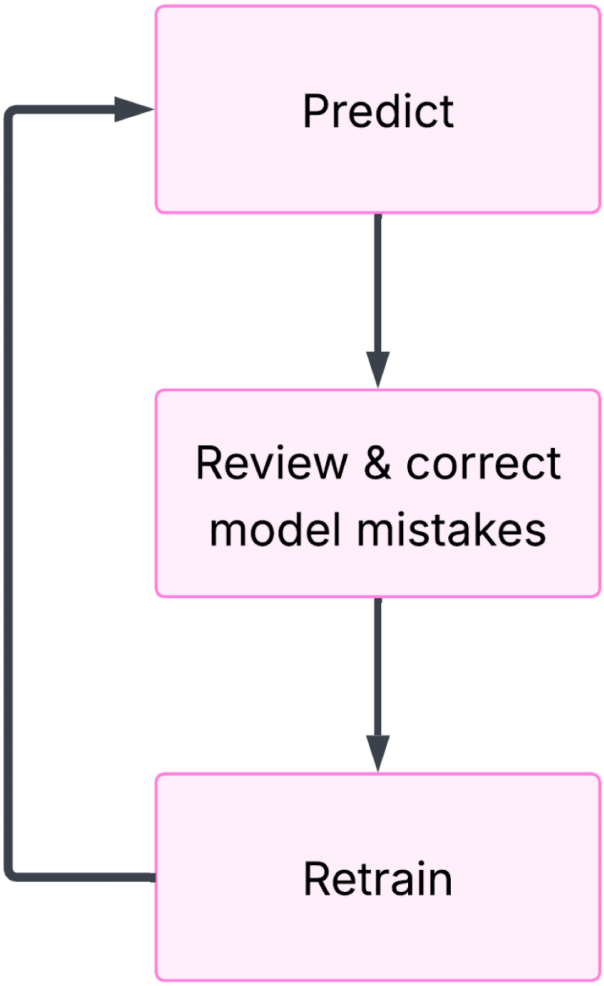


Figure 6.2: The three-step annotation cycle for efficient fine-tuning. Starting with a base model, you predict on new samples, manually review and correct any errors, then retrain the model on the corrected labels. This cycle repeats until the model achieves the desired performance level, making annotation much faster than labeling from scratch.

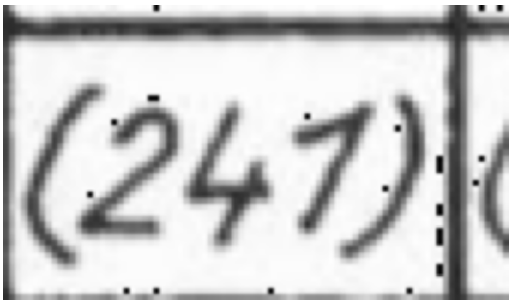


Figure 6.3: An example of a critical labeling error. The image shows a parenthesis "(" but is incorrectly labeled as a bracket. When such mislabeled examples are included in the training data, the model learns incorrect patterns that can degrade its performance on correctly labeled examples. Source: Fine-Tuning VLMs for Document Understanding.

Chapter 7

Processing Documents at Scale

7.1 Cost Implications

Warning

Costs can escalate quickly when processing millions of documents. A single misconfigured parameter can result in bills orders of magnitude higher than expected.

When processing at scale, VLMs introduce unique cost challenges:

- VLMs require ~10x more input tokens than text-based approaches
- High-resolution images (necessary for text reading) increase token counts further
- Processing millions of pages can quickly reach tens of thousands of dollars

7.2 Optimization Strategies

7.2.1 Maximize Prompt Caching

As discussed in Chapter 4, prompt caching reduces costs by 90% for cached tokens. At scale, this becomes critical:

- Ensure static prompts exceed 1024 tokens to activate caching
- Structure all prompts identically across documents
- Batch similar documents together to maximize cache hits

7.2.2 Right-Size Your Approach

Not all documents require VLMs:

1. **Text-only documents:** Use OCR + LLM (10x cheaper)
2. **Simple layouts:** Use traditional OCR + regex
3. **Complex visuals:** Use VLMs only when necessary

7.2.3 Resolution Optimization

Find the minimum viable resolution for your task:

- Test at 300 DPI, 450 DPI, and 600 DPI
- Use lowest resolution that maintains acceptable accuracy
- Different document types may need different resolutions

7.2.4 Hierarchical Processing

Implement answer-dependent processing:

1. Start with first 10 pages at medium resolution
2. If information not found, process more pages
3. If text unreadable, increase resolution
4. Only escalate when necessary

Most tasks complete at the lowest tier, dramatically reducing average cost.

7.3 Infrastructure Considerations

7.3.1 Essential Components

- **Parallel processing:** Process multiple documents simultaneously
- **Error handling:** Retry logic for API failures, rate limits
- **Progress tracking:** Monitor completion rates and identify bottlenecks
- **Quality assurance:** Sample validation to catch systematic errors
- **Cost monitoring:** Real-time spend tracking with alerts

7.3.2 Open Source vs API

Using APIs (OpenAI, Anthropic, Google):

+ No infrastructure management
+ Easier to get started
– Ongoing per-token costs
– Less control over processing

Self-hosting open source (Qwen, LLaVA):

+ Fixed infrastructure costs
+ Complete control and privacy
– Requires GPU infrastructure (A100+)
– More complex to maintain

For millions of documents, self-hosting often becomes cost-effective despite higher upfront complexity.

7.4 Best Practices

1. **Start small:** Process 100-1000 documents first
2. **Measure everything:** Track cost per document, accuracy, processing time
3. **Establish baselines:** Know your expected costs before scaling
4. **Implement monitoring:** Set up alerts for anomalous spending
5. **Scale gradually:** 10x at a time, validating quality at each step
6. **Validate continuously:** Sample output quality throughout processing

Key Point

The cost of fixing errors after processing millions of documents far exceeds the cost of thorough validation on smaller batches.

Chapter 8

Conclusion and Best Practices

8.1 Key Takeaways

Vision Language Models have fundamentally changed document understanding. The key insights from this book:

1. **VLMs are necessary when visual position matters:** If text meaning depends on layout, checkboxes, or visual context, VLMs are required
2. **Choose the right approach for each task:**
   - Regex for highly structured, consistent data
   - OCR + LLM for most text extraction needs
   - VLMs when visual information is critical
3. **Optimization is essential:** Proper prompt structure, caching, and model selection can reduce costs by 10x while improving quality
4. **Data inspection is invaluable:** Understanding your data deeply enables better prompts, annotation strategies, and fine-tuning decisions
5. **Fine-tuning works when done carefully:** Label quality, learning rate, and data balance are critical for success
6. **Scale gradually:** Start small, measure thoroughly, validate continuously

8.2 Recommended Workflow

For a new document understanding project:

1. **Manually inspect 100+ documents** to understand characteristics and challenges
2. **Start with simplest approach:** Try OCR + LLM first unless obviously visual
3. **Prototype and measure:** Test on small batch (100-1000 documents), track cost/accuracy/speed

4. **Optimize prompts:** Apply the four techniques from Chapter 4
5. **Benchmark models:** Test multiple providers (Anthropic, OpenAI, Google)
6. **Consider fine-tuning:** If base models underperform and you have volume to justify it
7. **Scale gradually:** 10x increases with validation at each step

8.3 The Future of VLMs

The field is rapidly evolving:

- **Improved efficiency:** Models are getting better at compressing visual information into fewer tokens
- **Longer contexts:** Context windows expanding to handle full documents
- **Better accuracy:** Particularly on handwriting and complex layouts
- **Lower costs:** Competition and optimization reducing API prices
- **Specialized models:** Domain-specific VLMs for medical, legal, technical documents

Key Point

VLMs will become increasingly important as we move beyond pure text to truly multi-modal AI systems.

8.4 Final Thoughts

Document understanding with VLMs is no longer experimental—it's production-ready technology processing millions of documents daily. Success requires:

- Understanding when VLMs are necessary vs nice-to-have
- Rigorous cost optimization and monitoring
- Careful validation and quality assurance
- Continuous learning as models and capabilities evolve

The techniques in this book are battle-tested on real production systems. Apply them systematically, measure results, and iterate based on your specific use case.

The future of document understanding is visual, contextual, and intelligent. VLMs are the key to unlocking that future.

Appendix A

Resources

A.1 Referenced Articles

This book is based on content from the following Towards Data Science articles by Eivind Kjosbakken:

1. **How to Apply Vision Language Models to Long Documents**
2. **4 Techniques to Optimize Your LLM Prompts for Cost, Latency and Performance**
3. **How to Consistently Extract Metadata from Complex Documents**
4. **How to Use Frontier Vision LLMs: Qwen-VL**
5. **Using Vision Language Models to Process Millions of Documents**
6. **Fine-Tuning VLMs for Document Understanding**