Phase 4: Multimodal Embeddings → CLIP/SigLIP models (text to image alignment)

Understanding the concept

A multimodal deep learning model called CLIP (Contrastive Language Image Pretraining) is used to understand and synchronize text and images in a shared semantic space. Text and images are not treated as distinct types of data by CLIP. Rather, it uses equal dimensional mathematical vectors to represent each of these domains in order to identify links between them.

Context related text and image representations should correlate to similar vectors, according to a basic tenet of CLIP. For example, the text "electric wiring layout" and an image of a machine wiring diagram will be mapped to nearby points in vector space. In systems that must simultaneously reason about textual and visual data, this skill is essential.

Why is CLIP required in multimodal RAG?

Traditional RAG systems work only with text. They read paragraphs, sentences, and keywords, and then try to find the best text-based answer. This works fine for articles or blogs, but technical manuals are very different. They don't just explain things with words, they depend heavily on diagrams, wiring layouts, warning symbols, step-by-step images, and visual instructions. If a system ignores those visuals, it's missing a huge part of the information.

This is where CLIP changes things.

CLIP understands both images and text in the same "language." Because of that:

- When a user asks a question in text, the system can now find relevant images that match the meaning of the question, not just the words.
- If an image is important, like a diagram or schematic, the system can also retrieve the supporting text that explains it.
- Most importantly, the visual context is not lost. The system understands *what the image represents*, not just that an image exists.

By combining text and visuals in the retrieval process, the RAG system becomes multimodal instead of text-only. This makes a big difference for technical and instruction-based questions, where seeing a diagram or warning symbol can be just as important as reading an explanation. As a result, the answers become clearer, more accurate, and much closer to how humans actually understand manuals.

How does CLIP Works?

CLIP consists of two encoders:

1. **Text Encoder** → Converts text into embeddings
2. **Image Encoder** → converts images into embeddings

Both encoders are trained jointly using contrastive learning. During training, the model learns to maximize similarity between matching image and text pairs and then maximize similarity between unrelated pairs. As a result, both text and images exist in a shared vector space, enabling direct similarity comparison.

Limitations and Considerations

While powerful, it's important to understand what CLIP can and cannot do.

First, CLIP does not actually read text inside images. In the case of a diagram, where small labels, numbers, or instructions may be written inside the image, CLIP does not understand those words directly. It only understands the overall meaning or visual concept of the image. For detailed text inside images, an OCR step is needed to extract the text before passing it to the system.

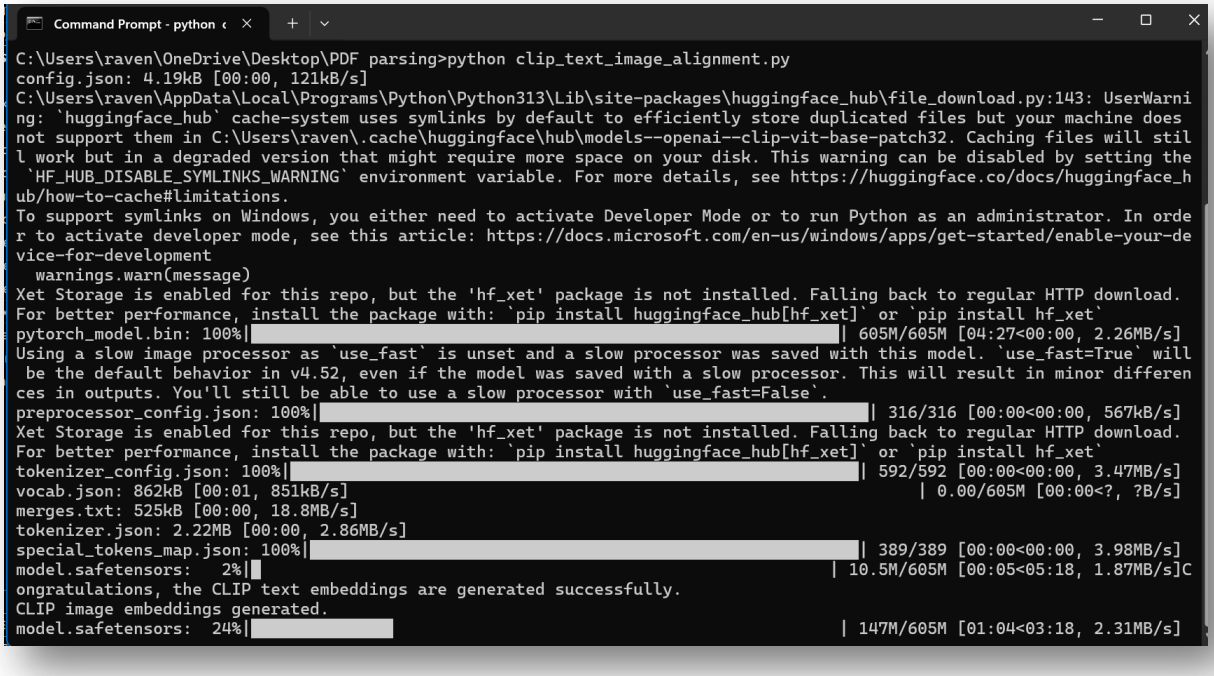
Second, image quality plays a big role in performance. Clear, high-resolution images with proper contrast work much better than blurry scans, low-quality photos, or heavily compressed diagrams. If the quality of the image is not good, then CLIP may not understand what the image really represents, which might affect retrieval accuracy.

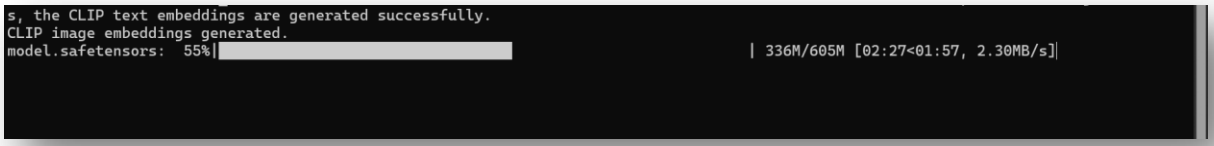
Another consideration is that of storage and computational cost: CLIP creates fairly large embeddings both for text and images. When you are working with thousands of manual pages, diagrams, and images, these embeddings take up more storage space and are more memory-consuming during retrieval compared to the text-only systems.

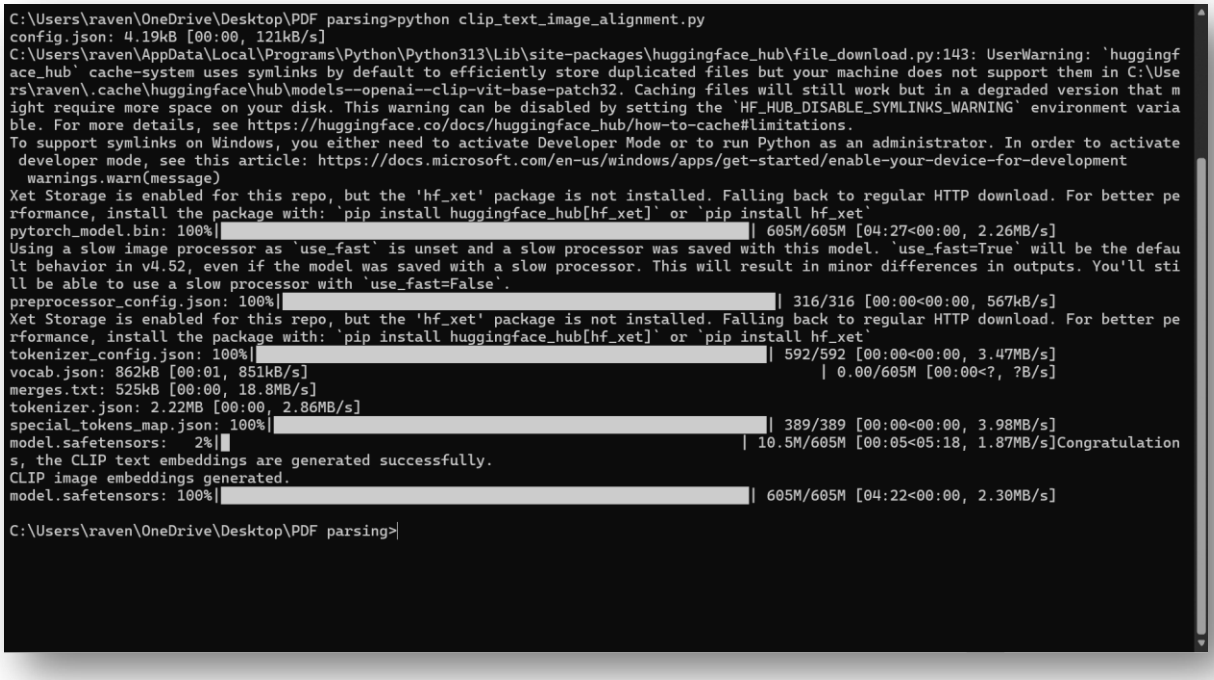
Third and lastly, domain-specific content might be challenging. General-purpose CLIP models are usually trained on general internet data. Sometimes technical manuals have symbols, engineering diagrams, medical illustrations, or specific visuals from a sector that CLIP is not able to understand out-of-the-box; therefore, fine-tuning or domain adaptation of the models significantly increases the accuracy.

Despite these limitations, CLIP presents an excellent balance of capability and simplicity. It enables developers to extend the multimodal understanding within a RAG pipeline without having to build intricate custom vision models. In most technical and instructional use cases, CLIP will show strong performance with relatively low implementation effort. Hence, in many practical multimodal RAG systems, CLIP would be an effective choice.

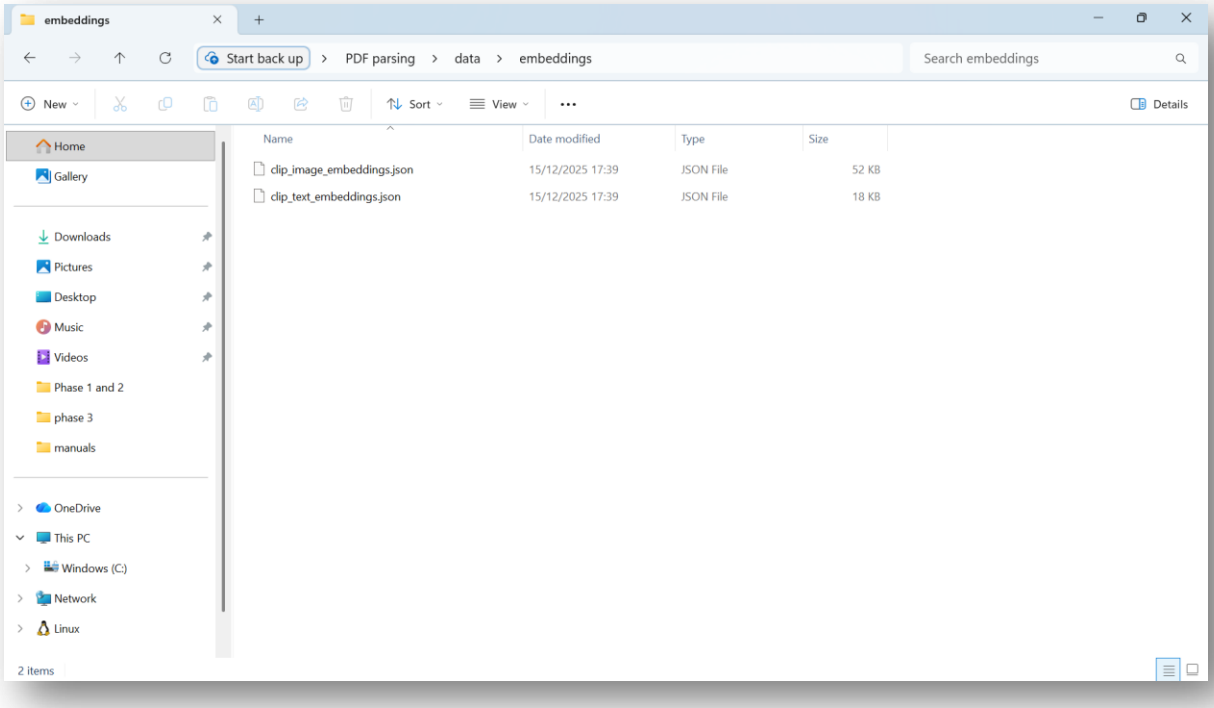
Testing




s, the CLIP text embeddings are generated successfully.
CLIP image embeddings generated.
model.safetensors: 55%| | 336M/605M [02:27<01:57, 2.30MB/s]


C:\Users\raven\OneDrive\Desktop\PDF parsing>python clip_text_image_alignment.py
config.json: 4.19kB [00:00, 121kB/s]
C:\Users\raven\AppData\Local\Programs\Python\Python313\Lib\site-packages\huggingface_hub\file_download.py:143: UserWarning: `huggingface_hub` cache-system uses symlinks by default to efficiently store duplicated files but your machine does not support them in C:\Users\raven\.cache\huggingface\hub\models--openai--clip-vit-base-patch32. Caching files will still work but in a degraded version that might require more space on your disk. This warning can be disabled by setting the `HF_HUB_DISABLE_SYMLINKS_WARNING` environment variable. For more details, see https://huggingface.co/docs/huggingface_hub/how-to-cache#limitations.
To support symlinks on Windows, you either need to activate Developer Mode or to run Python as an administrator. In order to activate developer mode, see this article: https://docs.microsoft.com/en-us/windows/apps/get-started/enable-your-device-for-development
warnings.warn(message)
Xet Storage is enabled for this repo, but the 'hf_xet' package is not installed. Falling back to regular HTTP download. For better performance, install the package with: `pip install huggingface_hub[hf_xet]` or `pip install hf_xet`
pytorch_model.bin: 100%| | 605M/605M [04:27<00:00, 2.26MB/s]
Using a slow image processor as `use_fast` is unset and a slow processor was saved with this model. `use_fast=True` will be the default behavior in v4.52, even if the model was saved with a slow processor. This will result in minor differences in outputs. You'll still be able to use a slow processor with `use_fast=False`.
preprocessor_config.json: 100%| | 316/316 [00:00<00:00, 567kB/s]
Xet Storage is enabled for this repo, but the 'hf_xet' package is not installed. Falling back to regular HTTP download. For better performance, install the package with: `pip install huggingface_hub[hf_xet]` or `pip install hf_xet`
tokenizer_config.json: 100%| | 592/592 [00:00<00:00, 3.47MB/s]
vocab.json: 862kB [00:01, 851kB/s] | 0.00/605M [00:00<?, ?B/s]
merges.txt: 525kB [00:00, 18.8MB/s]
tokenizer.json: 2.22MB [00:00, 2.86MB/s]
special_tokens_map.json: 100%| | 389/389 [00:00<00:00, 3.98MB/s]
model.safetensors: 2%| | 10.5M/605M [00:05<05:18, 1.87MB/s]Congratulations, the CLIP text embeddings are generated successfully.
CLIP image embeddings generated.
model.safetensors: 100%| | 605M/605M [04:22<00:00, 2.30MB/s]

C:\Users\raven\OneDrive\Desktop\PDF parsing>



About the implementation

The CLIP model was implemented to generate aligned vector representations for both textual chunks and extracted images. These embeddings reside in a shared vector space, enabling cross-modal understanding and forming the foundation for multimodal retrieval in later phases.