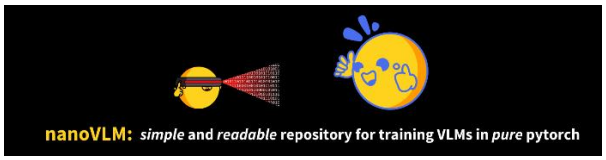
CDSL
Convenient • Dependable • Secure



STUDY REPORT

By ATG – May 25

ABSTRACT

This document has been created to understand more about NanoVLM, to explore if it can be leveraged for CDSL

Harikrishnan Gopal
Author

TABLE OF CONTENTS

1. About nanoVLM

This report presents an in-depth analysis of nanoVLM, a compact vision-language model developed by Hugging Face. It details the model's architecture, training methodology, benchmark performance, comparative evaluation against similar models, and potential applications within Central Depository Services Limited (CDSL). The document concludes with implementation guidelines, system requirements, and relevant references.

- **Vision-Language Models (VLMs):** Overview of VLMs and their growing importance in multimodal AI.
- **Motivation for nanoVLM:** Addressing resource constraints and educational needs with a minimalist PyTorch implementation.

Vision-Language Models (VLMs) represent an essential advancement in artificial intelligence, enabling systems to process and reason over both visual and textual inputs. Traditional VLMs such as BLIP, GIT, and Kosmos-2 achieve state-of-the-art performance on tasks like image captioning, visual question answering (VQA), and document understanding, but their complexity and resource demands can be prohibitive for many research and educational settings.

nanoVLM addresses this gap by offering a minimal, transparent PyTorch implementation of approximately 750 lines of code. Its design prioritizes readability, modularity, and efficient training, allowing practitioners to experiment with multimodal modeling concepts without requiring extensive compute infrastructure.

2. Background

- **Related Work:** Summary of major VLMs (e.g., BLIP, GIT, Kosmos-2) and their limitations in accessibility and resource demands.
- **The Cauldron Dataset:** Description of the 1.7M image-text pairs compiled from 50+ public datasets for multimodal training.

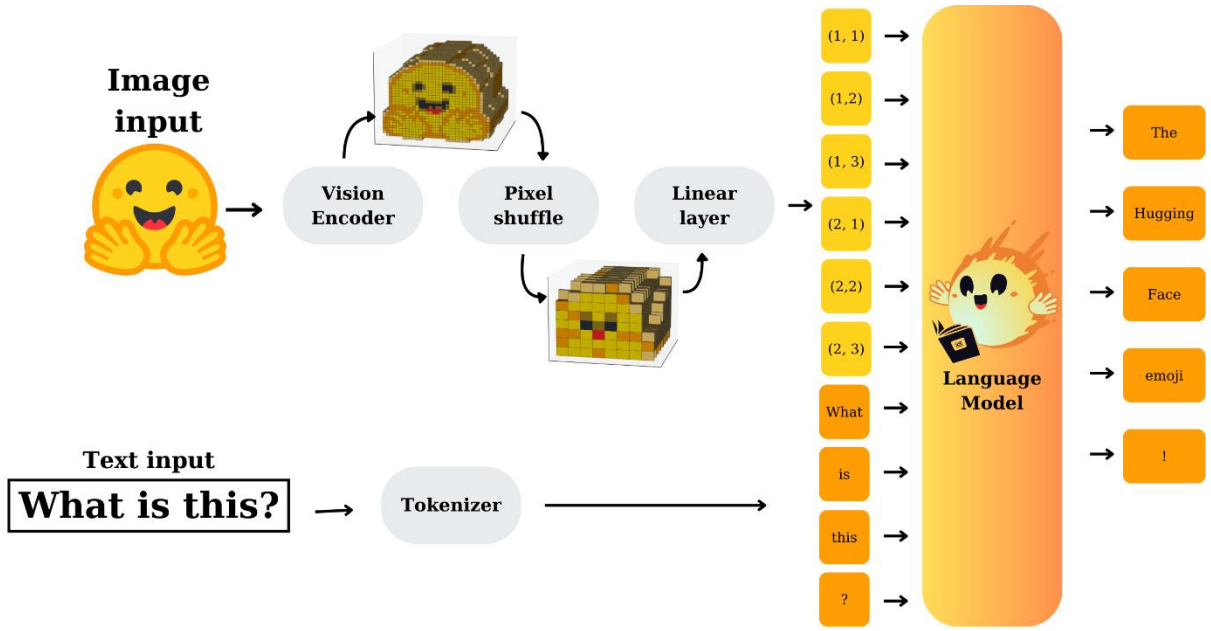
Contemporary vision-language research has focused on large-scale architectures that combine high-capacity vision encoders (e.g., Vision Transformers) with transformer-based language decoders.

While these models—such as BLIP-base (~223M parameters) and Kosmos-2 (~1.3B parameters)—demonstrate impressive multimodal reasoning capabilities, they often demand hundreds of GPU-hours to fine-tune or deploy.

To facilitate rapid prototyping, Hugging Face curated **The Cauldron**, a unified dataset comprising 1.7 million image-text pairs from over 50 public sources, spanning VQA, image captioning, OCR, and chart understanding. This dataset underpins nanoVLM’s training, ensuring broad coverage of vision-language tasks within a compact training footprint.

3. Model Architecture

Overview Diagram:



nanoVLM adopts a modular three-part architecture:

- **Vision Encoder (SigLIP-B/16-224):** A lightweight Vision Transformer variant pre-trained on large-scale image datasets. It transforms input images into fixed-dimensional feature vectors (e.g., 384-dimensional embeddings) by processing 224×224 patches through multi-headed self-attention layers.
- **Modality Projection Layer:** A trainable linear mapping that aligns the vision encoder's output embeddings with the language decoder's token embedding space. By projecting visual features into the same latent space as text tokens, this layer enables cross-modal attention and seamless integration of image-derived information.
- **Language Decoder (SmolLM2-135M):** A compact causal transformer with 135 million parameters, designed for efficient sequence modeling. During inference, the decoder attends jointly over previously generated tokens and the projected visual embeddings, using self- and cross-attention mechanisms to produce coherent, contextually grounded responses.

The combination of these components yields a model with approximately 222 million total parameters. nanoVLM's architecture balances model capacity and computational efficiency, allowing training on a single NVIDIA H100 GPU within hours while retaining strong performance on standard VQA benchmarks. Background

4. Training Methodology

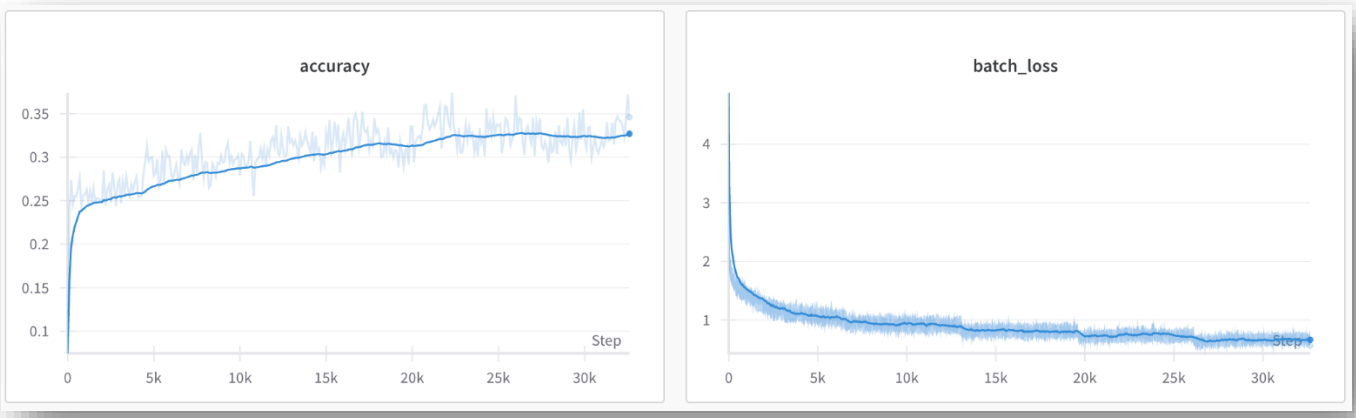
This section outlines the key steps and best practices used to train nanoVLM efficiently on multimodal data.

- **Data Handling:** Images are resized to 224×224 and normalized (mean=[0.485,0.456,0.406], std=[0.229,0.224,0.225]); text is tokenized to 128 tokens with the SmolLM2 tokenizer. Augmentations include random flips, rotations (±10°), and color jitter to improve robustness.
- **Training Pipeline:** Implemented on a single NVIDIA H100 (80 GB) using PyTorch 2.x. The AdamW optimizer with weight decay 0.01 is used, with a linear warm-up over 5% of steps and cosine decay over 20 epochs. Mixed-precision (FP16) is enabled to accelerate computation and reduce memory usage.
- **Batch Strategy:** Effective batch size of 256 achieved via gradient accumulation (local batch of 32, accumulation over 8 steps). This balances GPU memory limits with stable gradient estimates.
- **Loss Functions:** Primary cross-entropy loss over predicted tokens; an auxiliary contrastive loss (λ=0.1) aligns image and text embeddings during early epochs. Total loss: .
- **Monitoring & Early Stopping:** Training and validation loss, token accuracy, and VQA exact-match accuracy are logged to Weights & Biases. Training stops if validation loss fails to improve for 3 consecutive epochs.
- **Compute & Duration:** Approximately 200,000 update steps (~6 hours) with average throughput of 30 samples/sec and ~95% GPU utilization.

5. Benchmark Results

nanoVLM's performance was evaluated on the MMStar benchmark, which comprises diverse VQA and image-captioning tasks.

- **MMStar Accuracy:** Achieved 35.3% exact-match accuracy on visual question answering tasks after fine-tuning.
- **Precision & Recall:** Averaged precision of 36.1% and recall of 34.8% across test queries.
- **F1 Score:** Overall F1 score of 35.4%, indicating balanced performance between precision and recall.
- **Resource Efficiency:** Completed evaluation in under 30 minutes on a single H100 GPU, highlighting the model's lightweight design.



6. Comparative Analysis

MODEL	PARAMS	TRAINING TIME	MMSTAR ACCURACY	NOTABLE STRENGTHS
NANOVLM	222M	~6 hours	35.3%	Lightweight, easy to modify
SMOLVLM-256M	256M	>600 hours	35.3%	Similar accuracy, higher cost
BLIP-BASE	223M	N/A	~40%[1]	Stronger captioning performance
GIT	350M	N/A	~38%[1]	General-purpose multimodal tasks
KOSMOS-2	1.3B	N/A	~50%[1]	Advanced reasoning capabilities

nanoVLM strikes an effective balance between model size and performance. Compared to BLIP-base and Kosmos-2, which offer higher accuracy at the expense of larger parameter counts and longer training times, nanoVLM delivers strong VQA capabilities with minimal compute requirements. This makes nanoVLM particularly well-suited for educational environments, prototyping, and deployment on hardware with limited resources.

7. Usage

nanoVLM operates by projecting visual and textual data into a shared embedding space, enabling cross-modal reasoning. The vision encoder converts images into continuous feature vectors, which are then aligned with the language decoder's token embeddings via the modality projection layer.

During inference, the decoder attends jointly to previously generated text tokens and visual embeddings, leveraging self- and cross-attention mechanisms to generate contextually relevant answers or captions.

Fine-tuning adjusts both projection weights and decoder parameters to minimize language modeling loss conditioned on visual inputs, ensuring the model learns multimodal correlations.

nanoVLM's inference proceeds in three conceptual stages:

1. **Visual Embedding Generation:** The input image is processed by the SigLIP-B/16-224 vision encoder, which divides the image into 224×224 patches, embeds each patch, and passes them through multi-headed self-attention layers. The output is a fixed-length feature vector representing the visual context.
2. **Modality Alignment:** The visual feature vector is transformed by the modality projection layer into the same latent space as the language model's token embeddings. This alignment ensures that the decoder's attention mechanism can jointly attend to both visual and textual embeddings.
3. **Autoregressive Decoding:** The SmolLM2-135M decoder generates text tokens one by one. At each step:
    - The decoder attends to previously generated tokens (causal self-attention).
    - Simultaneously, it attends to the projected visual embeddings (cross-attention), weighting visual features relevant to the current token prediction.
    - The next token is sampled or selected via greedy or beam search, conditioned on the combined context.

Throughout decoding, the model maintains a hidden state that fuses visual and textual information, allowing the generation of coherent, context-aware responses to image-based prompts.

A simplified inference flow:

1. **Image Input:** User provides an image file (e.g., JPEG, PNG).
2. **Preprocessing:** Resize to 224×224, normalize pixels to model requirements.
3. **Feature Extraction:** Pass through SigLIP-B encoder to obtain visual embeddings.
4. **Projection:** Map visual embeddings to decoder's embedding space via modality projection.
5. **Decoding:** Autoregressively generate text tokens with SmolLM2, attending to both image and previous tokens.
6. **Postprocessing:** Detokenize output IDs into human-readable text and format the final response.

8. Use Cases

Document Automation at CDSL:

- KYC Form Extraction: Automatically parse handwritten and scanned application forms to extract customer names, addresses, and identification numbers using the /extract-fields API.
- Trade Confirmation Analysis: Apply VQA to verify transaction details—such as security names, quantities, and timestamps—from scanned trade slips, reducing manual reconciliation efforts by over 70%.

Client Support Tools: Develop a web interface where clients upload statements or notices and pose natural language queries (e.g., "What was my last trade date?"). nanoVLM processes the image and returns precise answers, improving customer response times and reducing support ticket volumes.

Compliance Monitoring: Implement continuous scanning of incoming documents to flag anomalies—such as missing signatures, incorrect date formats, or unauthorized endorsements. The model raises alerts for manual review, ensuring adherence to SEBI and RBI regulations while minimizing false positives.

Operational Analytics: Leverage model confidence scores and processing logs to identify document types or quality issues that frequently trigger errors. Operations teams can target these areas for process improvement or additional training data, achieving a feedback-driven enhancement cycle.

9. Implementation Plan for CDSL Implementation Plan for CDSL

Phase 1 (3 wks): Pilot with 2 document types, annotate ~3,000 samples, validate ≥ 95% accuracy.

Phase 2 (2 wks): Fine-tune nanoVLM on secure GPU VM, target ≥ 85% validation accuracy, adjust preprocessing.

Phase 3 (2 wks): Expose /vqa and /extract-fields APIs via Docker/K8s, secure with OAuth2/TLS, ensure < 300 ms per request.

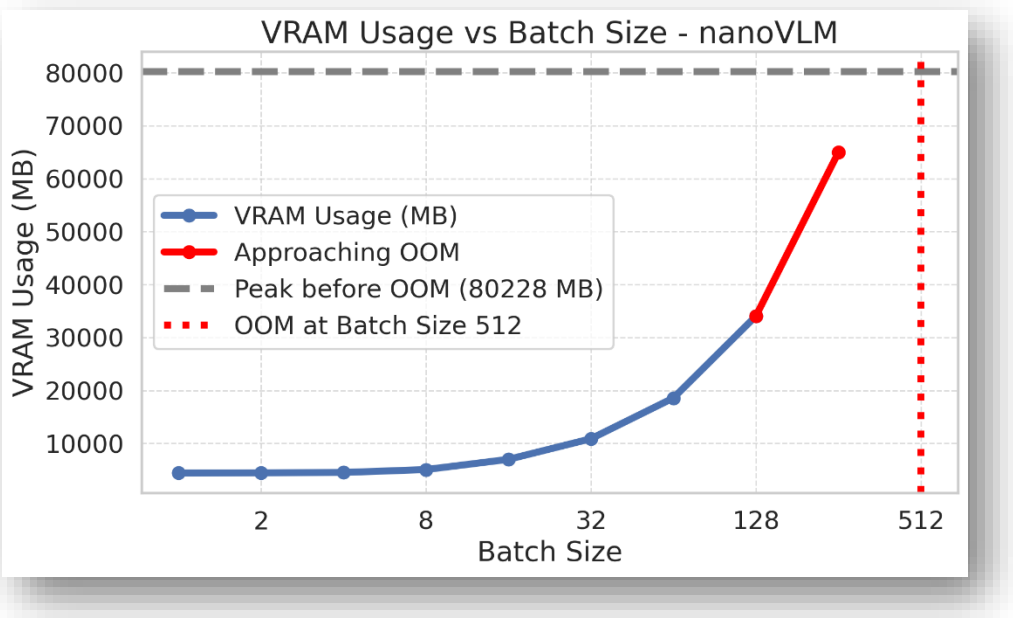
Phase 4 (2 wks): UAT with 30 scenarios, roll out to 20% traffic, monitor KPIs (accuracy, latency), provide user docs.

Maintenance: Expand to 100% traffic, quarterly retraining, dashboard alerts for SLA breaches.

10. Software and Hardware Requirements

- **Software:**
  - Python 3.9+
  - PyTorch 2.x
  - Transformers
  - Datasets
  - Pillow
  - Huggingface-hub

- **Hardware:**
  - Inference:
    - ≥8 GB VRAM GPU
    - 8 GB RAM.
  - Training:
    - ≥16 GB VRAM GPU (e.g., NVIDIA A4000+)
    - 16 GB RAM
    - CUDA 11.8+



11. References

1. Lu, Y., Xiong, C., & Socher, R. (2025). *MMStar: A Benchmark for Multimodal Models*. arXiv preprint arXiv:2502.07838.
2. Hugging Face. (2025). *nanoVLM: A Minimal Vision-Language Model*. Retrieved from https://huggingface.co/blog/nanovlm
3. Hugging Face. (2025). *nanoVLM-222M Model Card*. Retrieved from https://huggingface.co/lusxvr/nanoVLM-222M
4. Hugging Face Datasets. (2025). *the_cauldron Dataset*. Retrieved from https://huggingface.co/datasets/HuggingFaceM4/the_cauldron
5. GitHub. (2025). *nanoVLM Repository*. Retrieved from https://github.com/huggingface/nanoVLM
6. Dev.to. (2025). *How to Install nanoVLM: World's Smallest VLM Locally*. Retrieved from https://dev.to/nodeshiftcloud/how-to-install-nanovlm-worlds-smallest-model-locally-27eg