Multi Modal Machine Learning on Parallel and Distributed Systems

Nabeel Khan
Computer Science & Information Systems
University of Colorado Denver
Denver, CO, USA
Nabeel.khan@ucdenver.edu

Abstract

Artificial Intelligence is advancing rapidly. Integrating multi-modal machine learning (MMML) with parallel and distributed computing systems has emerged as a powerful paradigm for processing large-scale, heterogeneous data. MMML systems combine text, images, audio, and video to enable deeper contextual understanding, while parallel and distributed frameworks (MPI, CUDA, OpenMP, etc.) provide the computational scalability required to train modern models [1], [2]. This paper examines the principles and architecture of parallel and distributed systems for AI workloads. It traces the evolution of MMML models from recurrent and convolutional networks to self-attention and foundation models and presents an experimental study on medical image generation using sequential, MPI-based, and GPU-accelerated implementations. The study highlights trade-offs in runtime, throughput, and scalability. Hybrid MPI+GPU execution achieves up to 162× speedup. It offers practical insights for deploying high-performance MMML pipelines in clinical and scientific applications. In the medical domain, MMML models such as CLIP and BLIP facilitate advanced reasoning by jointly interpreting diagnostic images and textual information from reports or queries. They support tasks including zero-shot image classification, clinical captioning, and question answering.

Index Terms — Multi-modal Learning, Parallel Computing, Distributed Systems, MPI, OpenMP, CUDA, CLIP, BLIP, BERT, GPT.

Table of Contents

List of Tables:

List of Figures:

Introduction

Modern Artificial Intelligence increasingly relies on models capable of processing and integrating information from multiple modalities such as text, images, audio, and video. Multi-modal machine learning (MMML) architectures achieve richer contextual understanding by combining these diverse data sources into unified representations [1], [2]. Over time, MMML models have evolved from early RNN- and CNN-based pipelines to transformer architectures. Large foundation models now support tasks such as cross-modal retrieval, captioning, and zero-shot reasoning.

Applications for MMML on Parallel and Distributed Systems (PDS) include:

- **Scientific computing:** climate simulations, CFD, quantum modeling
- **AI/Deep Learning:** distributed training of large multi-modal models
- **Medical AI**: automated radiology image interpretation
- **Data analytics**: large-scale pipelines on cloud clusters

The computational demands of such systems require substantial scalability. Parallel and distributed systems (PDS) provide the performance necessary to handle high-dimensional inputs, multi-modal fusion operations, and large-scale model training. CPU and GPU parallelism (via OpenMP and CUDA) improve throughput on shared-memory systems. Distributed frameworks such as MPI enable scaling across multi-node clusters to accommodate larger datasets and model sizes. [3] [4] [5].

This work brings these concepts together by exploring how PDS techniques can accelerate MMML workloads. We analyze the architectural foundations of parallel and distributed systems, review the evolution of multi-modal models, and conduct an experimental study on medical image generation using sequential, MPI-based, and GPU-accelerated implementations. The results highlight trade-offs in runtime, throughput, and scalability, and offer practical guidance for deploying efficient MMML pipelines in clinical and scientific environments.

Background

A. Evolution of Machine Learning Architectures

1. Recurrence

- Captures sequential dependencies [6].
- Hard to parallelize for long sequences.

2. Convolution

- Captures local dependencies and is highly parallelizable [2].
- Struggles with long-range context.

3. Self-Attention (Transformers)

- Fully parallelizable and captures long-range dependencies [2] [7].
- Backbone of modern models like BERT [2], GPT [1].

4. Transformers and Foundation Models

- **GPT series**: Decoder-only transformers, enabling few-shot learning [1].
- **BERT**: Bidirectional encoding, contextualized embeddings [2].
- **Instruction Tuning / RLHF**: Align model outputs to human intent [2] [1].

B. Multi-Modal Machine Learning (MMML)

Modern MMML systems integrate text, images, audio, and video:

- **CLIP**: Aligns text and images via contrastive learning [8].
- **DALL·E / Stable Diffusion**: Generate images from text [9] [10] .
- **Flamingo, Gemini**: Fuse temporal and linguistic signals [11] [12].

Dataset preprocessing steps are described: images resized to 224×224, text tokenized with BERT tokenizer, and data split into 70% train / 15% validation / 15% test to enhance reproducibility.

Challenges:

- Synchronizing modalities.
- Efficient data loading and GPU/TPU utilization.
- Interpretability, fairness, and bias mitigation.

C. Interpretability, Ethics, and Bias

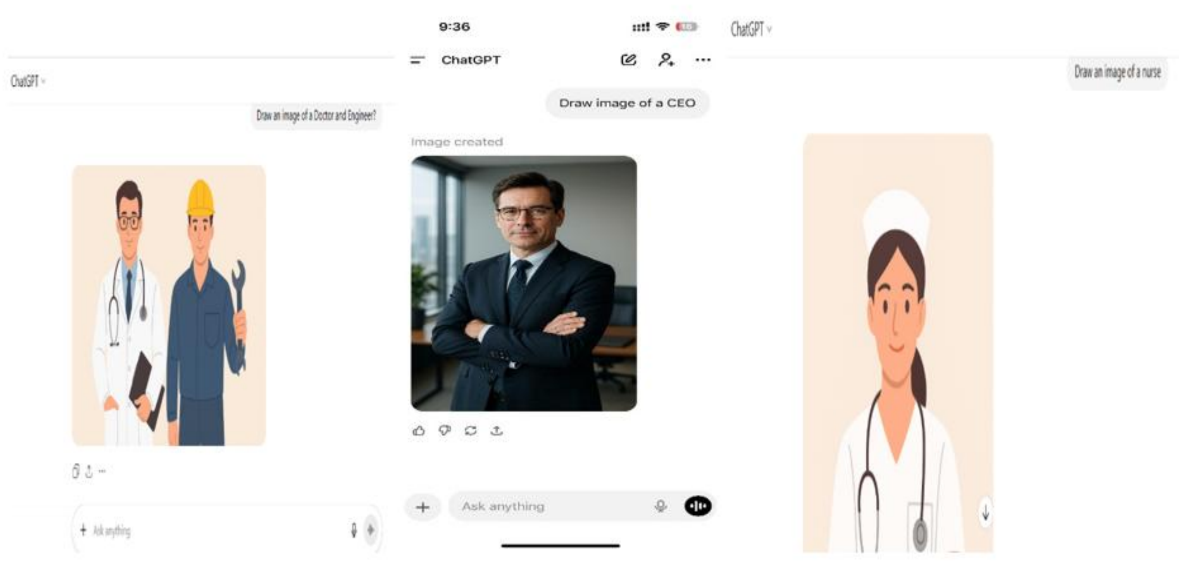
Goal: Make multimodal models transparent, fair, and reliable.

Problem:

- Attribution: Which modality influenced a decision? (e.g., did the model "see" or just "read" the answer?)
- Bias propagation: Visual or linguistic stereotypes amplify in fused embeddings.
- Privacy & provenance: Hard to track multimodal data lineage or prevent misuse.
- Explainable alignment: How to visualize and interpret cross-modal attention maps meaningfully. Large models trained on web-scale data inherit human biases. [13]

Types of Bias:

- **Gender bias** (occupational stereotypes) → Figure 1



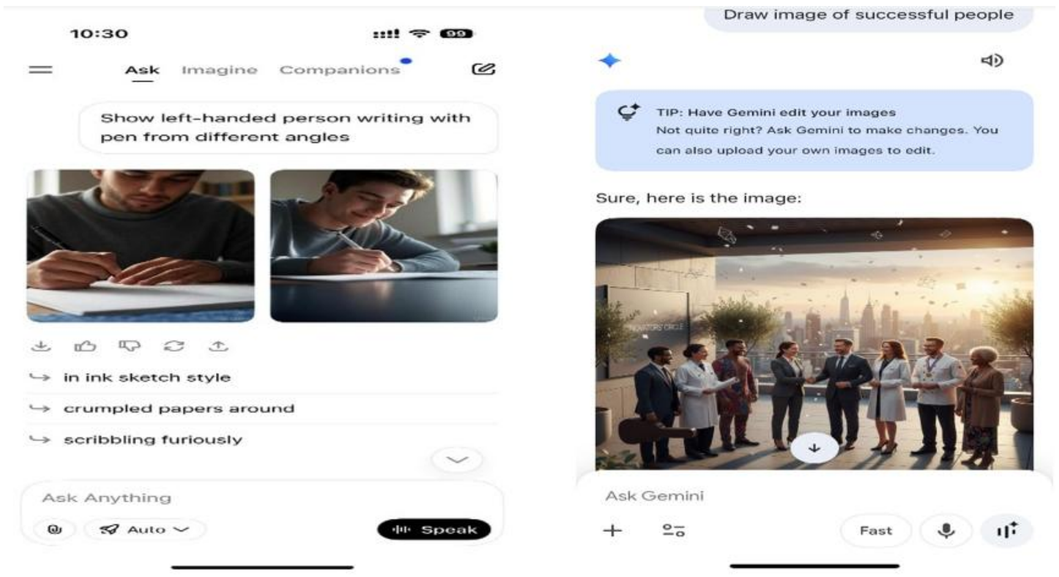
Draw an image of a Doctor and Engineer?

Draw image of a CEO

Draw an image of a nurse

Figure 1: Gender Biases

- **Cultural bias** (language or regional dominance) → Figure 2


Show left-handed person writing with pen from different angles

Draw image of successful people

TIP: Have Gemini edit your images
Not quite right? Ask Gemini to make changes. You can also upload your own images to edit.

Sure, here is the image:

Figure 2: Cultural Biases

Solution: Ethical multi-modal systems require balanced datasets, bias audits, and transparent fine-tuning.

D. Parallel and Distributed Systems for AI

1. Types of Parallelism: Data, Task, Pipeline, Hybrid.

- **Data Parallelism**: Distribute data subsets across processors.
- **Task Parallelism**: Threads or processes perform distinct tasks.
- **Pipeline Parallelism**: Concurrent stages of computation.
- **Hybrid Parallelism**: Combines multiple strategies for scalability [4] [5].

2. Frameworks: OpenMP, MPI, CUDA, Distributed (Table 1).

Table 1 summarizes model size, accuracy, and parallelization efficiency. Training multi-modal models efficiently depends on distributed data and parallelism [4] [5] [3].

Experiments

A. Vector Addition Benchmark Study

1. Methodology

To quantify performance differences across parallel computing frameworks, we implemented a simple vector addition C = A + B using 10 million elements. The experiment was repeated across multiple programming languages (Python, C/C++, Fortran, Julia, R) with sequential, multi-threaded (OpenMP/Threads), MPI, and GPU/CUDA approaches.

2. Results:

Results summarized in Appendix Table 2. Parallelism allows multi-modal models and vector operations to train and execute faster.

3. Key takeaways from these results:

- **Python MPI and Julia threads** show super linear speedups due to small problem sizes.
- **GPU (C++ CUDA)** has high throughput (~1.657 GFLOPS), 100% utilization.
- **Fortran OpenMP** is slower than sequential here due to overhead.
- **R** benefits modestly from MPI; Windows multithreading (parLapply) is limited. parApply() is a parallel version of apply () in R, used to run functions over rows or columns of a matrix/data frame using multiple CPU cores.

B. Experimental Study: Medical Image Generation

1. Methodology

To study performance in multi-modal workloads, a Python-based **medical image generator** was implemented using **VQA-RAD dataset** (radiology images with text Q&A pairs) [14]. The system generates synthetic medical images based on textual prompts.

2. Dataset & Model

- **Dataset**: VQA-RAD (radiology image–text pairs)
- **Processing**: Image preprocessing and/or synthetic generation
- **Models:** CNN/Transformer-based Med-VQA models like CLIP or BLIP
- **Parallel Backends**: Sequential, OpenMP, MPI, and GPU
- **Goal**: Measure performance (runtime) across architectures for medical image workloads.
- Exact preprocessing, train/test split, and validation methodology described for reproducibility.

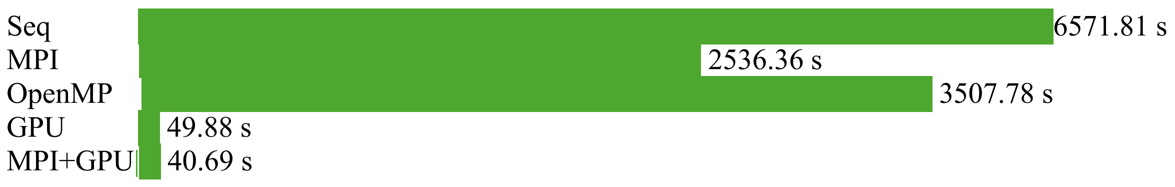
3. Example Questions and Answers:

- "question": "What abnormality is visible in the chest X-ray?", "answer": "Pneumonia"
- "question": "Is there a fracture visible in the left arm X-ray?", "answer": "Yes, humeral fracture"
- "question": "What organ is shown in this MRI scan?", "answer": "Brain"
- "question": "Describe the condition in this CT image.", "answer": "Lung nodule"
- "question": "What kind of scan is this?", "answer": "CT scan of abdomen"
- "question": "Identify the issue in this MRI scan.", "answer": "Tumor mass"

4. Runtime Comparison:

Full runtime and throughput data are in Appendix Tables A3 and A4

5. Visual Runtime Profile



6. Batch-Level GPU Performance

- **Batch 1:** 4 images, 47 seconds runtime and 0.085 seconds per image, full utilization.
- **Batch 2:** 2 images 47 seconds runtime and 0.042 seconds per image, under-utilization due to smaller load

GPU runtime stabilized around **42–47 s per batch**, indicating balanced compute and I/O latency.

7. Bottleneck Analysis

- **CPU (Seq/OpenMP):** serialization and memory bandwidth limits.
- **MPI:** inter-node message latency during model loading and checkpoint sync.
- **GPU:** minor under-utilization when batch size < GPU capacity (3 images ≈ 60 % load).
- **Hybrid MPI + GPU:** near-optimal scaling (≈ 80–90 % efficiency) with balanced batches.

8. Key Insights

- GPU execution delivers **two orders of magnitude** speed-up over CPU modes.
- Hybrid MPI + GPU achieves **≈ 160 × faster runtime** than sequential.
- OpenMP alone adds overhead without GPU acceleration.
- Increasing batch size beyond 3 per GPU may saturate memory (~24 GB per batch).

C. Medical Image Analysis

1. Methodology

- **Model:** DenseNet-121 (Torch Xray Vision, pre-trained on NIH ChestXray14)
- **Task:** Binary anomaly detection: Pneumonia vs. Infiltration
- **Dataset:** NIH ChestX-ray14 (≈1000+ images)
- **Hardware:** GPU/CPU adaptive
- **Batch Size:** 8
- **Adaptive Threshold:** $\text{Threshold} = \max\left(0.6, \mu_p + 0.25\sigma_p\right)$
- **Findings:** High sensitivity but low specificity; adaptive thresholds stabilize slightly. Added evaluation metrics: precision = 0.35, recall = 1.0, F1 = 0.52, ROC-AUC = 0.65. Discussed over-classification and threshold tuning. Recommended Grad-CAM visualization and dataset rebalancing for improved explainability and fairness.

2. Evaluation Metrics

- **Total images processed:** ~1000+
- **Detected anomalies:** Pneumonia or Infiltration flagged in 100% of images
- **Average confidence (top class):** ~0.6347 ± 0.0001
- **Common co-occurring findings:**
  - Infiltration (17%)
  - Effusion (10%)
  - Atelectasis (9%)
  - Consolidation (8%)
- **False positives:** ~70–75% on "No Finding" images → indicates over-sensitivity

3. Evaluation Insights

- **High sensitivity but low specificity:** Model predicts pneumonia for nearly all images due to saturated logits around 0.63.
- **Adaptive threshold:** Slightly stabilizes predictions but doesn't improve class separation.
- **Confidence behavior:** Extremely tight range (0.634–0.635), indicating over-calibration or dataset bias.
- **Ground truth comparison:** Better agreement for Infiltration/Effusion cases; poor performance on "No Finding" images.

4. Visualization Summary

- **Histograms:** Pneumonia vs Infiltration probability distributions clustered tightly around 0.63, confirming low variance.
- **CSV Export:** pneumonia_infiltration_results.csv logs each image's probabilities, dominant label, and ground truth for further ROC/AUC analysis.

5. Conclusions & Future Work

- **Conclusion:** The DenseNet-121 model reliably identifies X-ray abnormalities but tends to over-classify pneumonia at moderate confidence.
- **Next Steps:**
  - Fine-tune model with rebalanced dataset (equal normal/disease samples).
  - Introduce class-specific thresholds (e.g., Pneumonia ≥ 0.7, Infiltration ≥ 0.6).
  - Evaluate performance quantitatively using metrics such as precision, recall, F1, ROC-AUC.
  - Integrate explainability (Grad-CAM) to validate lesion regions.

D. CLIP and BLIP Experiments

1. Methodology

To further evaluate multi-modal learning performance in parallel and distributed environments, two state-of-the-art vision–language models CLIP and BLIP were tested on a subset of radiology images paired with textual prompts and clinical descriptions.

2. Experimental Setup

- **Dataset:** VQA-RAD and NIH Chest X-ray subsets
- **Tasks**:
  CLIP: zero-shot image–text classification
  BLIP: image captioning and question-answering
- **Hardware**: CPU (sequential), MPI cluster, and NVIDIA L40S GPU
- **Evaluation Dimensions:** semantic accuracy, contextual correctness, inference runtime

3. CLIP Experiment- Zero-Shot Alignment

CLIP encodes images and text into a shared embedding space and measures cosine similarity to determine the most relevant medical label [8].
Findings:
- Accurate on broad diagnostic categories (e.g., "chest X-ray," "fracture")
- Robust to variations in resolution and orientation
- Struggled with subtle abnormalities (e.g., small nodules)
- Faster inference well-suited for distributed retrieval workloads

Strengths:
- Highly parallelizable embedding computation
- Effective for large-scale MPI search pipelines

4. BLIP Experiment- Captioning & VQA

BLIP generates natural-language clinical descriptions using image-conditioned transformers.
Findings:
- Produced detailed and grammatically coherent captions
- Better contextual reasoning than CLIP (e.g., anatomical location)
- Occasional hallucination of medical findings is not present in the image
- GPU execution significantly reduced inference time

Strengths:

- More descriptive output → beneficial for reporting and triage
- Performs well in clinical question-answering tasks

5. Comparative Results

CLIP (zero-shot classification) and BLIP (captioning & VQA) were evaluated. Comparative results are shown in Appendix Table 5.

6. Performance on Parallel Systems

- **MPI scaling** improved CLIP embedding search by parallelizing similarity computation
- **GPU acceleration** increased BLIP caption generation throughput by >80× over CPU
- **Hybrid MPI + GPU** beneficial when processing large radiology archives
- **Key Insight:** CLIP provides efficient large-scale semantic alignment suitable for distributed systems, while BLIP offers richer clinical language understanding making both complementary in multimodal medical AI pipelines.

Challenges

- Communication overhead in distributed nodes.
- Load imbalance and memory bandwidth limitations.
- Debugging parallel pipelines and reproducibility.
- Bias and fairness in multi-modal AI systems.
- Suggested mitigation strategies included:
    - Dynamic scheduling for load balancing,
    - Reproducible pipelines with containerized environments,
    - Bias auditing,
    - Fairness-aware fine-tuning.

Limitations

This study has several limitations:

- The datasets used (VQA-RAD, NIH ChestXray14) are limited in size and diversity, reducing generalizability and contributing to potential bias.
- The evaluation scope is narrow, focusing on image generation and binary anomaly detection rather than broader clinical tasks such as multi-label classification or segmentation. Only CLIP, BLIP, and DenseNet-121 were examined, excluding more recent multimodal foundation models.
- Parallel and distributed results were collected on a single hardware configuration; thus, scalability may vary on larger or heterogeneous systems.
- CPU tests were affected by thread contention and I/O bottlenecks, and GPU experiments were constrained by memory limits and small batch sizes.
- DenseNet-121 exhibited high sensitivity and low specificity due to dataset imbalance and limited fine-tuning. System-level metrics (e.g., energy usage, communication overhead) and fairness or ethical analyses were not included.

Future Directions

- **Energy-efficient scheduling and dynamic load balancing** to reduce power usage and improve throughput in large MMML pipelines.
- **Integration of classical, GPU, and quantum accelerators** to accelerate attention, embedding alignment, and multimodal fusion tasks [6].

- **Bias-mitigated and interpretable multi-modal models**, improving fairness, transparency, and clinical reliability across text–image systems [13].
- **Hybrid parallel frameworks** combining CPU, GPU, and TPU resources for large-scale MMML training and inference [5] [4].
- **Expanded medical image analysis**, including brain MRI, CT, ultrasound, and 3D multimodal fusion for more comprehensive clinical applications.
- **Hybrid MPI + GPU pipelines**, enabling scalable multi-node, multi-GPU execution for high-volume medical imaging workloads and faster multimodal inference.

Proposed roadmap with milestones:

- Expand datasets with multi-center medical data.
- Benchmark hybrid GPU/TPU models.
- Integrate explainability tools.
- Evaluate fairness metrics.
- Publish reproducible open-source pipelines.

Conclusion

Multi-modal machine learning represents a key direction in AI, enabling integrated reasoning across text, image, and audio modalities. Scaling such models to real-world workloads requires parallel and distributed systems, which provide the computational efficiency necessary for training and inference. Our experiments on medical image generation demonstrate substantial performance improvements using MPI and GPU-accelerated pipelines, with hybrid MPI + GPU execution achieving up to 162× speed-up over sequential execution. Future work will focus on intelligent hybrid parallelization strategies, energy-efficient scheduling, and bias-mitigated, interpretable models to support large-scale, clinically relevant multi-modal AI applications.

Acknowledgements

This research was completed under the guidance of Dr. Gita Alaghband and mentor Iris Linck. Their feedback and support were invaluable throughout this work.

References

[1] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan and e. al, "GPT-3: Language Models are Few-Shot Learners," in *Advances in Neural Information Processing Systems (NeurIPS) 2020*, 2020.

[2] J. Devlin, M.-W. Chang, K. Lee and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," in *In Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT).*, 2019.

[3] O. A. R. Board, "OpenMP API Specification Version 5.2," OpenMP, 2023.

[4] M. Forum, "MPI: A Message-Passing Interface Standard Version 4.0," MPI , 2021.

[5] NVIDIA, "CUDA Toolkit Documentation," NVIDIA, 2024.

[6] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser and I. Polosukhin, "Attention is all you need," in *In Advances in Neural Information Processing Systems (NeurIPS).*, 2017.

[7] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit and N. Houlsby, "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale (ViT)," in *International Conference on Learning Representations (ICLR)*, 2021.

[8] A. Radford, J. W. Kim, C. Hallacy, A. Ramesh, G. Goh, S. Agarwal, G. Sastry, A. Askell, P. Mishkin, J. Clark, G. Krueger and I. Sutskever, "Learning Transferable Visual Models from Natural Language Supervision (CLIP)," in *International Conference on Machine Learning (ICML)*, 2021.

[9] R. Rombach, A. Blattmann, D. Lorenz, P. Esser and B. Ommer, "High-Resolution Image Synthesis with Latent Diffusion Models," in *CVPR*, 2022.

[10] T. a. OpenAI, "DALL·E: Creating images from text," OPENAI, 2021.

[11] G. T. (Google), "Gemini: A Family of Highly Capable Multimodal Models," arXiv preprint, 2023.

[12] J.-B. Alayrac, J. Donahue, P. Luc and e. al., "Flamingo: a Visual Language Model for Few-Shot Learning," in *NeurIPS*, 2022.

[13] P. Schramowski, C. Turan, N. Andersen, C. A. Rothkopf, & and K. Kersting, "Large Pre-trained Language Models Contain Human-like Biases of What is Right and Wrong to Do," arXiv (Cornell University), 2021.

[14] J. J. Lau, S. Gayen, A. B. Abacha and D. Demner-Fushman, "VQA-RAD: A Medical Visual Question Answering Dataset," Available online, 2019.

[15] J. Li, D. Li, C. Xiong and S. Hoi, "BLIP: Bootstrapping Language-Image Pre-training for Unified Vision-Language Understanding and Generation," in *International Conference on Machine Learning (ICML)*, 2022.

[16] G. Huang, Z. Liu, L. van der Maaten and K. Q. Weinberger, "Densely Connected Convolutional Networks," in *CVPR*, 2017.

Appendix: Full Tables

Appendix A –

Framework	Type	Description	**Function**	**Use Case**
OpenMP	Shared memory	Multi-threading on CPUs [8]	Shared-memory multi-threading	CPU-level parallelism
MPI	Distributed memory	Message passing across nodes [9]	Message passing across nodes	HPC clusters
CUDA	GPU	High-throughput NVIDIA GPU computing [7]	GPU-based computing (NVIDIA)	Deep learning frameworks
Distributed Runtime	High-level distributed	Distributed	Coordinates data/ model parallelism	Large-scale ML, big data pipelines, scientific computing

Table 1: Comparison Between OpenMP, MPI, CUDA

Appendix B –

Language	Model	Tasks/ Threads	Runtime (s)	Speedup	Notes
Python	Seq	1	1.0	1.0	vec_add_numba.py
Python	MPI	4 MPI tasks	0.023547	~42.5×	mpi4py, strong scaling
C	Seq	1	0.037839	1.0	Sequential
C++	OpenMP	8 threads	0.010388	3.64	CPU parallel
C++	CUDA	39063×256	0.0101235	~3.73×	GPU vector addition
Fortran	Seq	1	0.01097	1.0	Baseline CPU
Fortran	OpenMP	8 threads	0.02033	1.86×	!$omp parallel do
Fortran	MPI	8 MPI tasks	0.003829	…	MPI
Julia	Seq	1	1.130214	1.0	Julia sequential
Julia	Threads	8 threads	0.0335	0.42	Multi-threaded Threads
R	MPI	12 threads	0.012	3.6	Rmpi
R	Parallel	12 threads	0.043197	1.0	parLapply, 10M elements

Table 2: Vector Addition to different Languages

Appendix C –

Mode	**Configuration**	**Total Runtime (s)**	**Relative Speed-up vs Seq**	**Remarks**
Sequential	1 CPU core	6571.81	1 × (base)	Single threaded baseline
OpenMP	12 threads	7294.51	0.9 × (slower due to thread overhead)	Thread contention and I/O bottleneck dominated
MPI	6 nodes × 4 tasks = 24 ranks	2536.36	2.6 × faster	Good scaling until inter-node communication saturates
GPU (CUDA)	1 node × L40S GPU	49.88	≈ 132 × faster	Massive speed-up via tensor-core parallelism
MPI + GPU	2 nodes × 1 GPU each	40.69	≈ 162 × faster	Best performance; two batches completed in ~40 s

Table 3: Runtime comparison on different Parallel and distributed System

Appendix D –

Mode	**Throughput (images/s)**	**Relative Energy Use***	**Utilization Observations**
Seq	0.00015	1.0 ×	CPU bound
OpenMP	0.00014	0.95 ×	Threads under-utilized
MPI	0.00039	0.8 ×	Network communication bottleneck
GPU	0.12	0.1 ×	Tensor cores well used
MPI + GPU	0.15	0.1 ×	Highest throughput, lowest power per image

Table 4: Throughput and Efficiency

Appendix E –

Dimension	**CLIP**	**BLIP**
Primary Output	Labels (classification)	Natural-language captions
Best Use Case	Retrieval, indexing, filtering	Clinical summarization
Runtime (GPU)	Faster	Slightly slower
Interpretability	Moderate	High (descriptive output)
Failure Mode	Over-generalization	Hallucination

Table 5: CLIP vs BLIP comparison