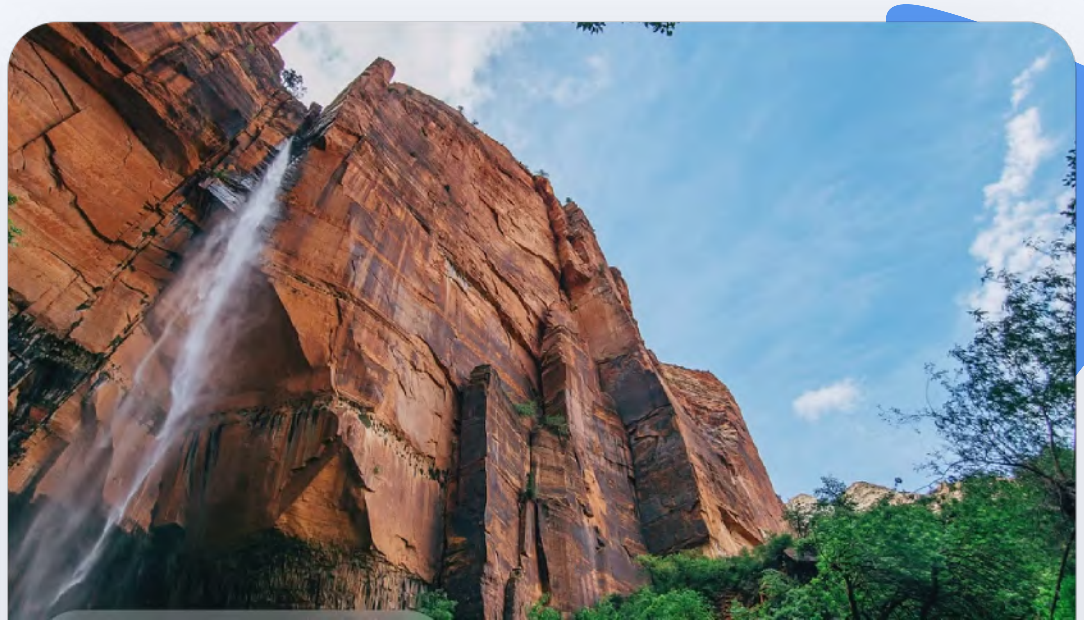
Parallelizing Text-to-Image Generation Using Diffusion

Team 21: Abdul Azeem Syed and Ramshankar Bhuvaneswaran
CSYE 7105 - Instructor - Dr. Handan Liu

Text



01

Motivation

Usually in text to image generation models, **GPUs are used in preprocessing** which includes VAE to convert the image into embeddings and CLIP model to generate text embeddings and attention masks from caption. As we know these **resources are quite expensive and not as easily accessible** as CPUs. So, our motivation for this project is to perform the pre-processing section of Text to Image generation mainly by **using multiple CPUs and try to check performance as GPU** resulting in more economically feasible results.

02

Dataset Overview

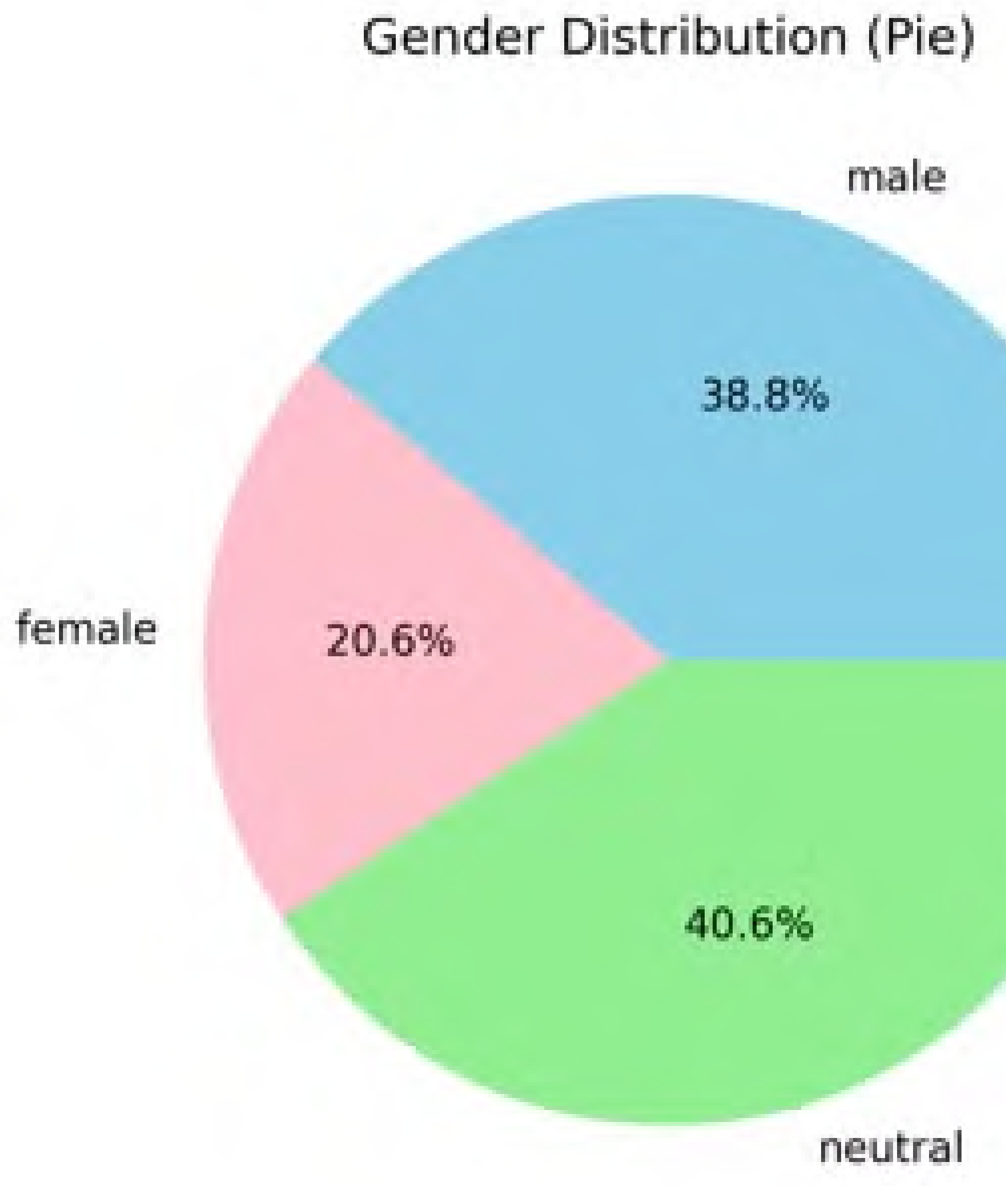
Dataset Overview

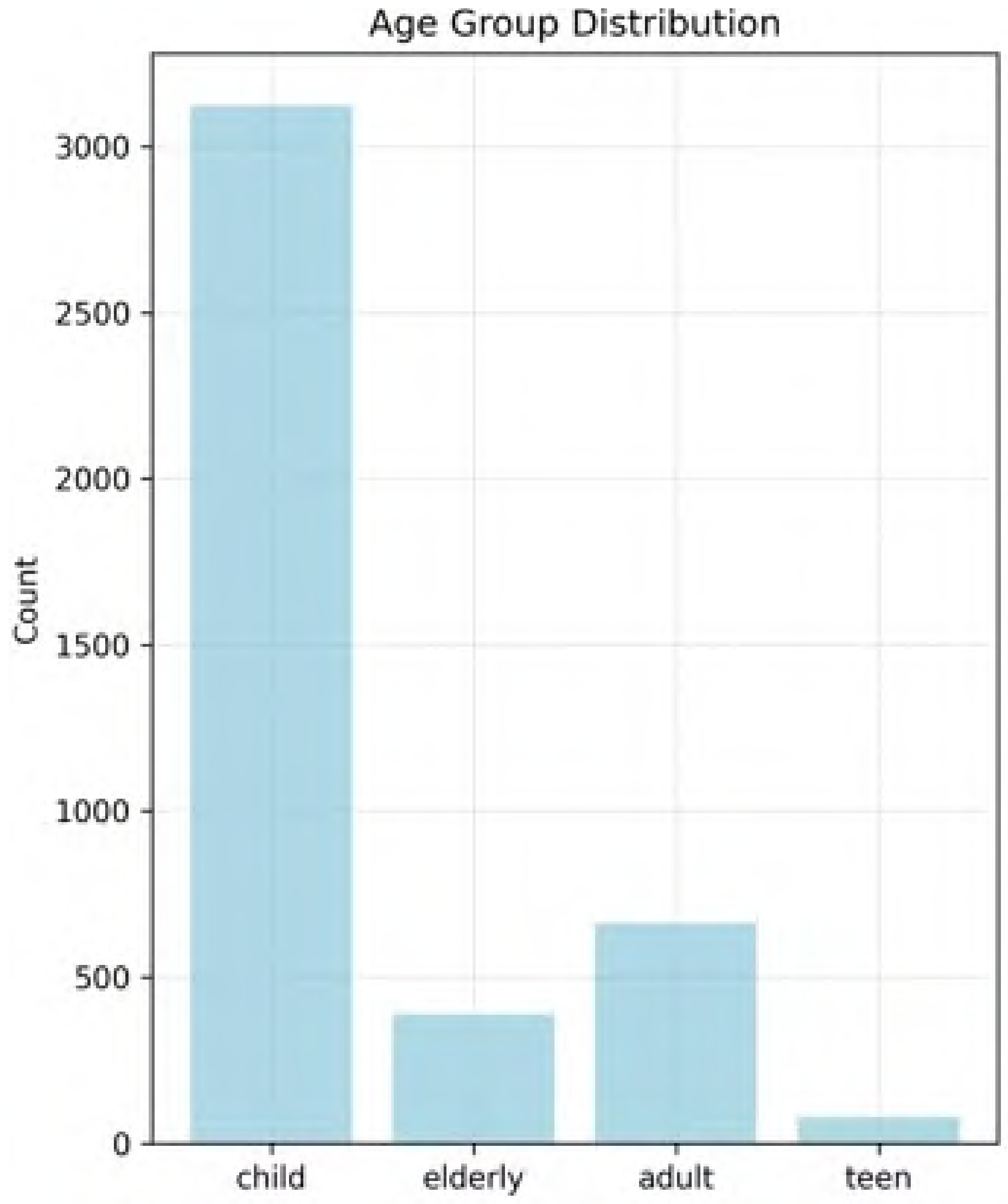
The MS COCO (Microsoft Common Objects in Context) dataset is a widely used benchmark for tasks such as object detection, segmentation, keypoint detection, and image captioning.

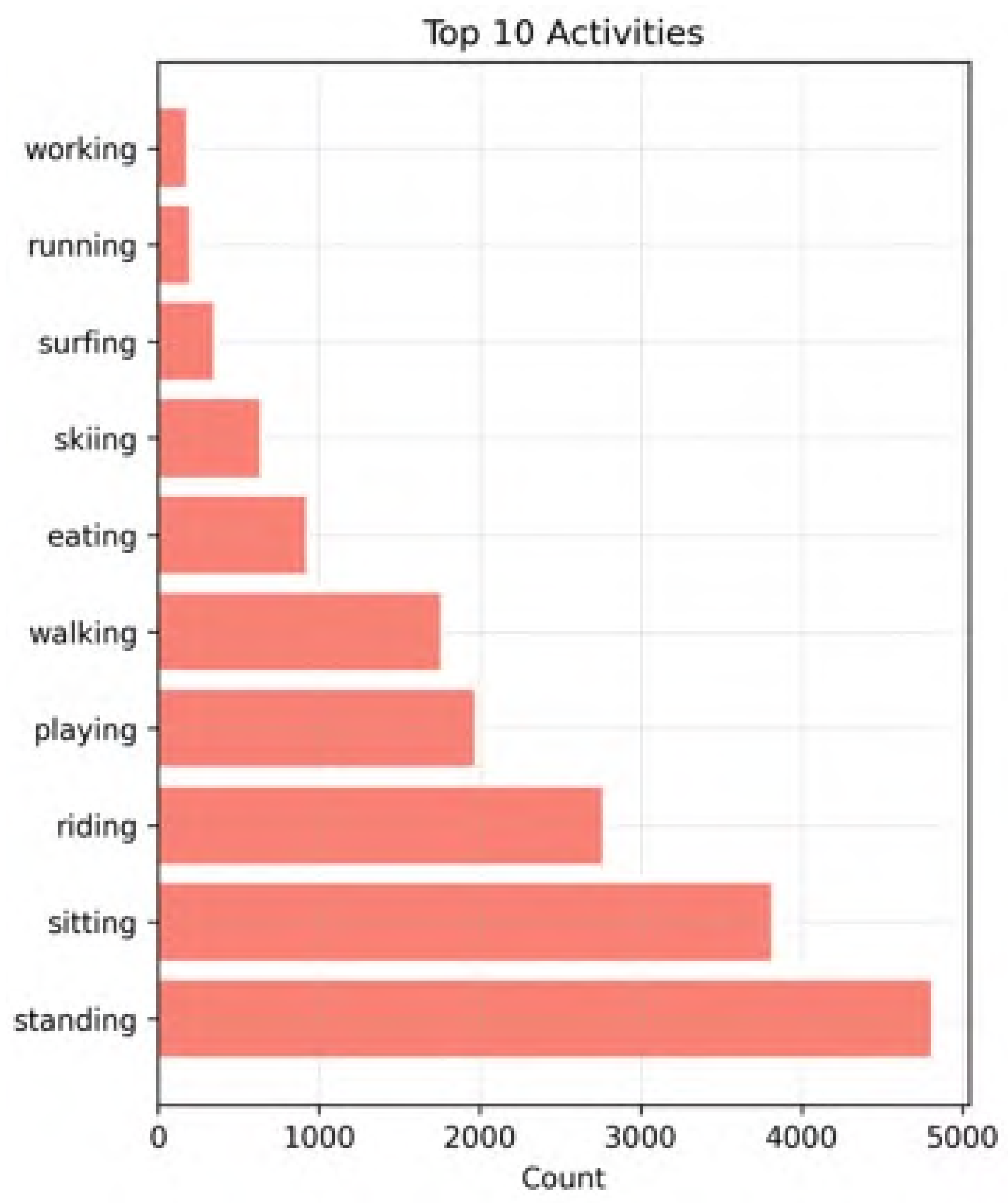
The train2014 subset of the dataset includes the following details:

- **Number of images: 83,000**
- **Total size: Approximately 13 GB**
- **Image resolution: 640×480 pixels**

83,000 Images → FILTERED → **48,339 Images**


Gender Distribution (Pie)
male
38.8%
female
20.6%
40.6%
neutral


Age Group Distribution
Count
3000
2500
2000
1500
1000
500
0
child
elderly
adult
teen



Dataset Demographics Summary:

Total images: 48339

Gender distribution: male: 14995 (31.02%)

female: 7940 (16.43%)

neutral: 15453 (31.97%)

Age group distribution: child: 3131 (6.48%)

elderly: 413 (0.85%)

adult: 608 (1.26%)

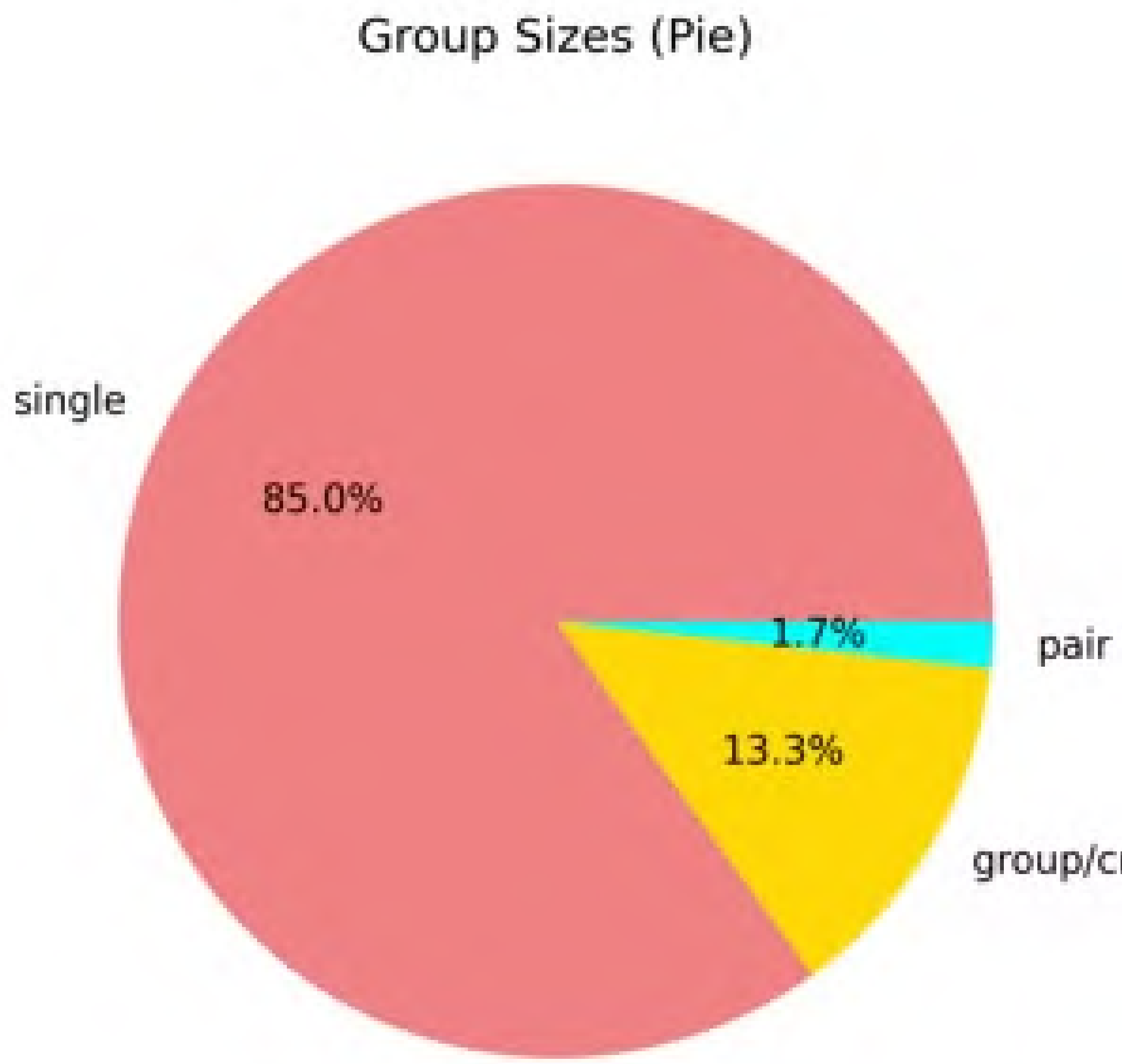
teen: 89 (0.18%)

Group size distribution:

single: 41146 (85.12%)

pair: 828 (1.71%)

group/crowd: 6365 (13.17%)



Try Pitch

03

Milestones

Milestones

- Filtering dataset
- Converting images into embedding using pretrained VAE
- Analysing the implimentation
- Converting captions into embedding using pretrained CLIP
- Analysing the implimentation
- Merging and training UNET with latent diffusion using noise scheduler
- Analysing the implimentation

04

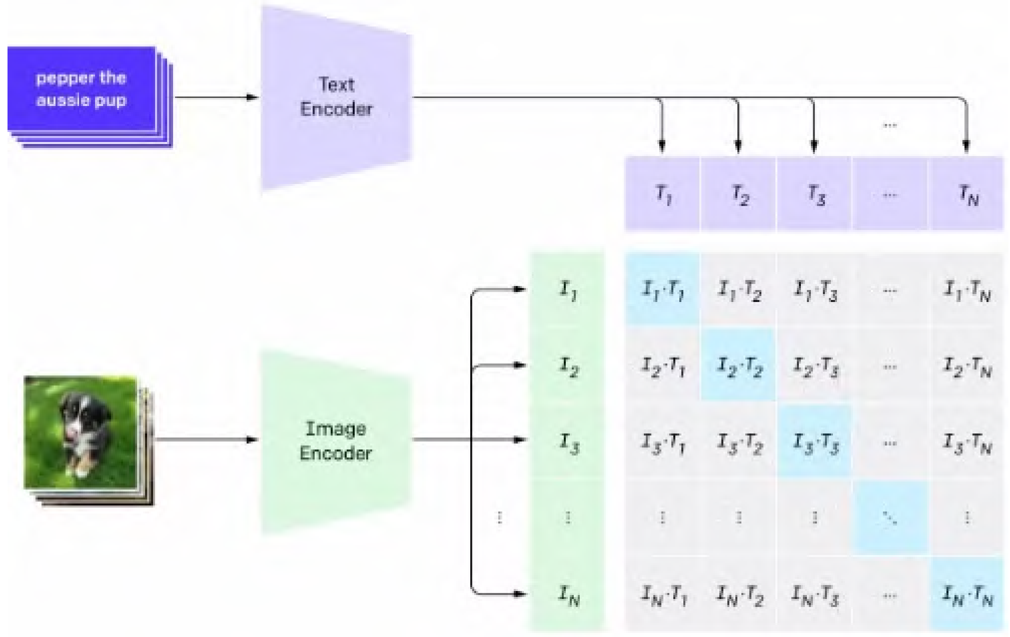
Process

Pre-processing

The pre processing consists of two main processes

VAE (Variational Autoencoder): This is used to embed images into **latent representations**, effectively reducing their dimensionality while retaining essential features for downstream tasks. These embeddings are a **compact representation** of the visual data, enabling efficient processing and training.

CLIP (Contrastive Language–Image Pretraining): This model is used to generate **embeddings for text captions and perform attention masking**. The attention masks help focus on relevant tokens in the captions, ensuring the model processes meaningful parts of the input text while ignoring irrelevant details.



Multi-Processing

Parallel Processing Methods

- **CPU-Based Parallelism:**
  - Conducted experiments with **1, 2, 4, and 8 CPUs**.
  - Implemented **two multiprocessing techniques**:
    1. **Native Multiprocessing** (Python `multiprocessing` library).
    2. **Joblib** for optimized workload distribution.

  - Analyzed performance to identify the more efficient approach.

- **GPU-Based Parallelism**:
  - Conducted experiments with **1, 2, 3, and 4 GPUs**.
  - Leveraged **multiprocessing** to maximize GPU utilization for encoding tasks.

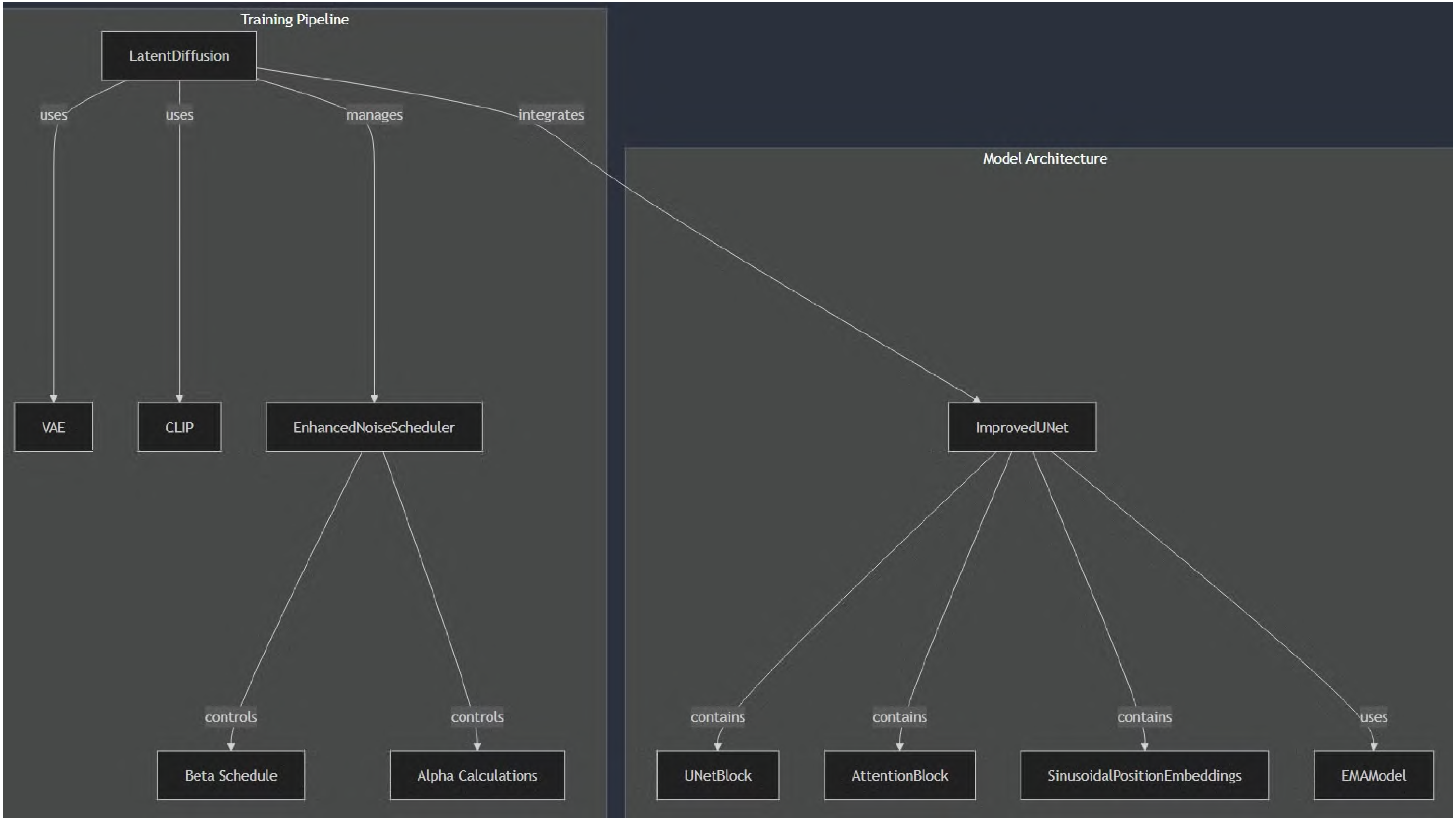
Training the model

Once embeddings were generated, we integrated them into a diffusion training pipeline using a **U-Net-based architecture**. The training loop utilized **Distributed Data Parallelism** to ensure load balancing and synchronization across multiple computational nodes.

Flow:
1. Data Loading → 2. Model Initialization → 3. Distributed Setup(DDP) →4. Training Loop (Noise Addition → Prediction → Loss Calculation → Update) →5. Evaluation and Checkpointing

Key Features:
- Multi-GPU training support
- Mixed precision training
- Gradient scaling
- Memory optimization
- Exponential Model Average

We ran 50 Epochs as the highest which gets the loss of 0.3133


Training Pipeline
LatentDiffusion
uses
uses
manages
integrates
VAE
CLIP
EnhancedNoiseScheduler
controls
controls
Beta Schedule
Alpha Calculations
Model Architecture
ImprovedUNet
contains
contains
contains
uses
UNetBlock
AttentionBlock
SinusoidalPositionEmbeddings
EMAModel

Try Pitch

Inference (testing the model)

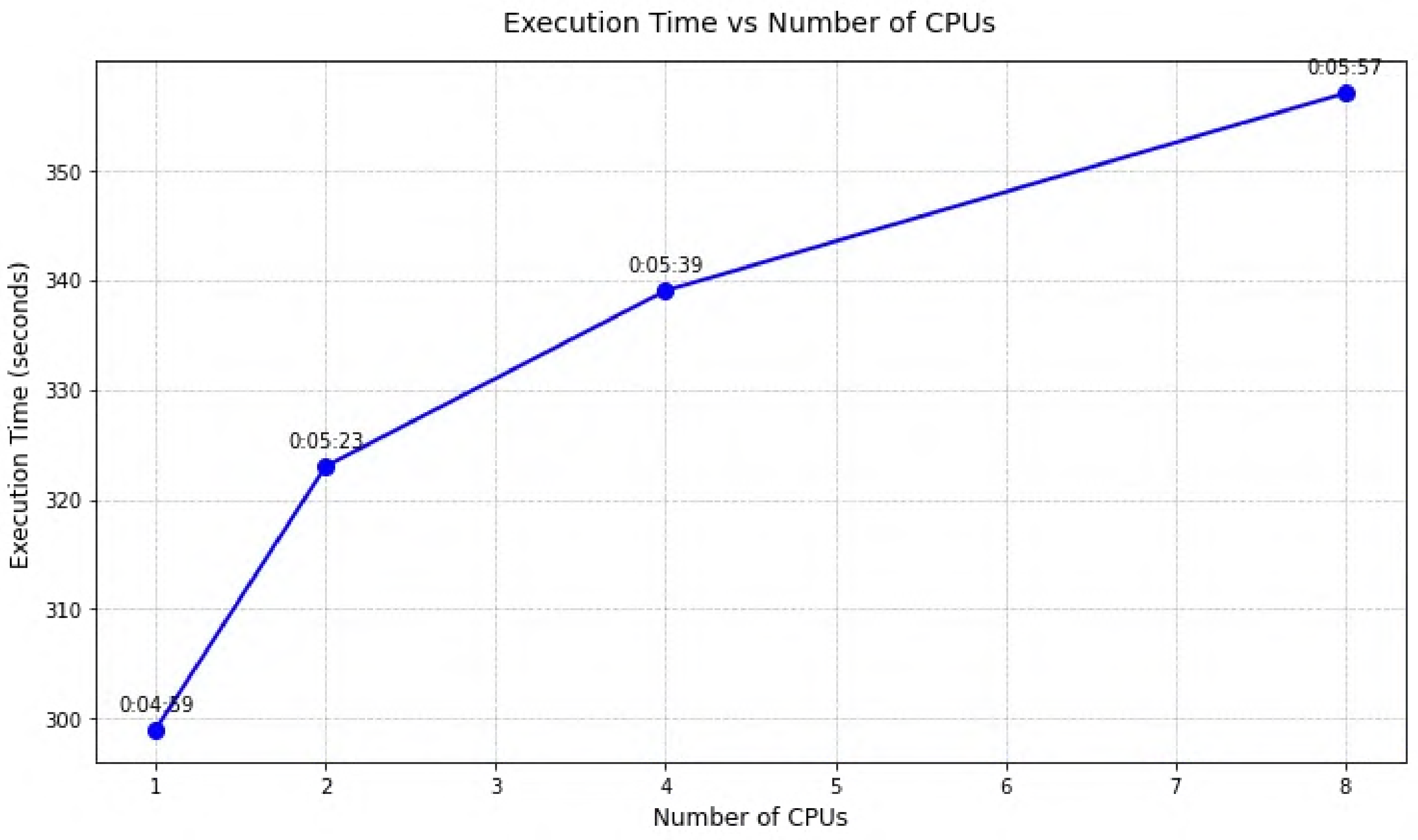
A man in cycle

05

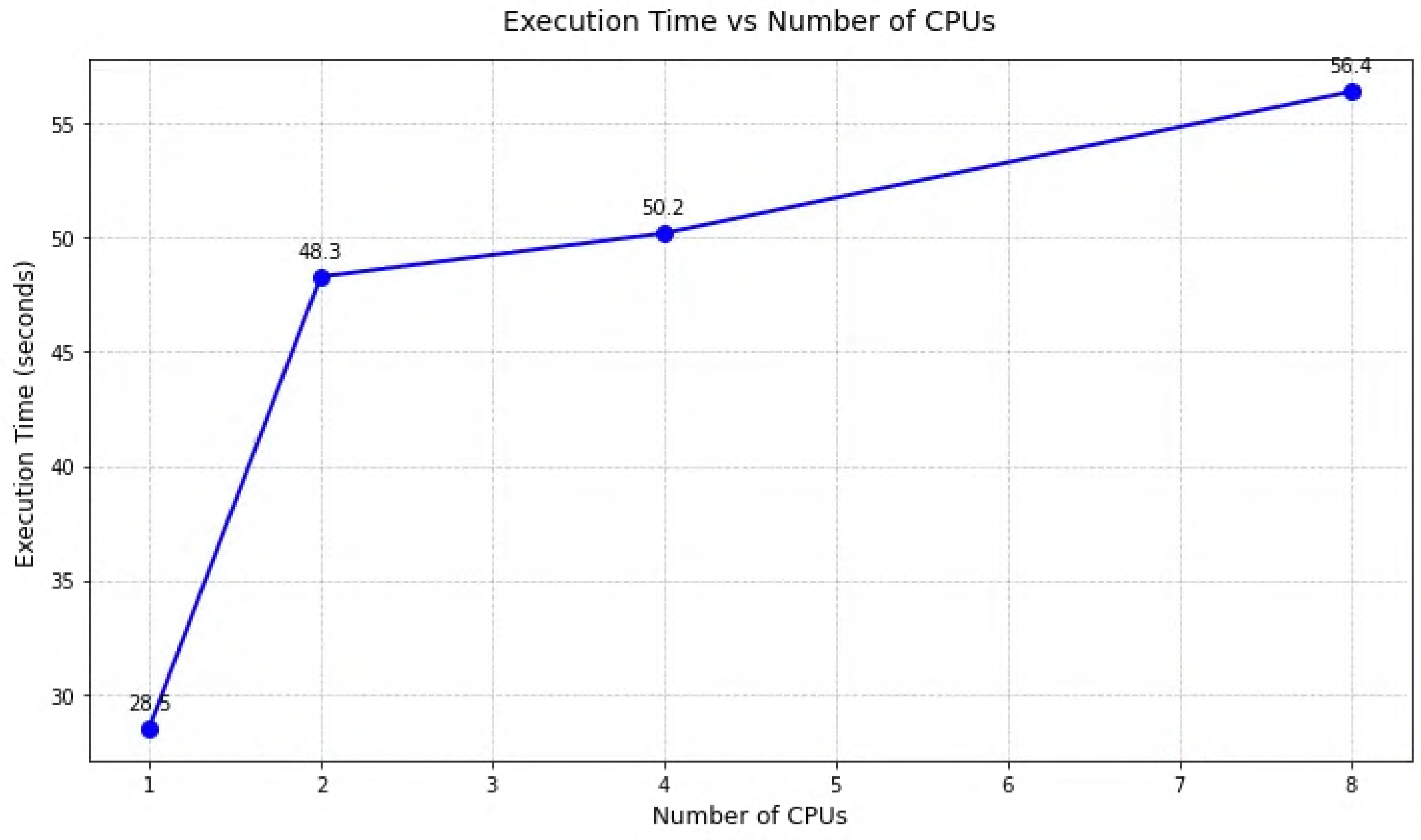
Analysis

Native Multiprocessing in CPUs

VAE

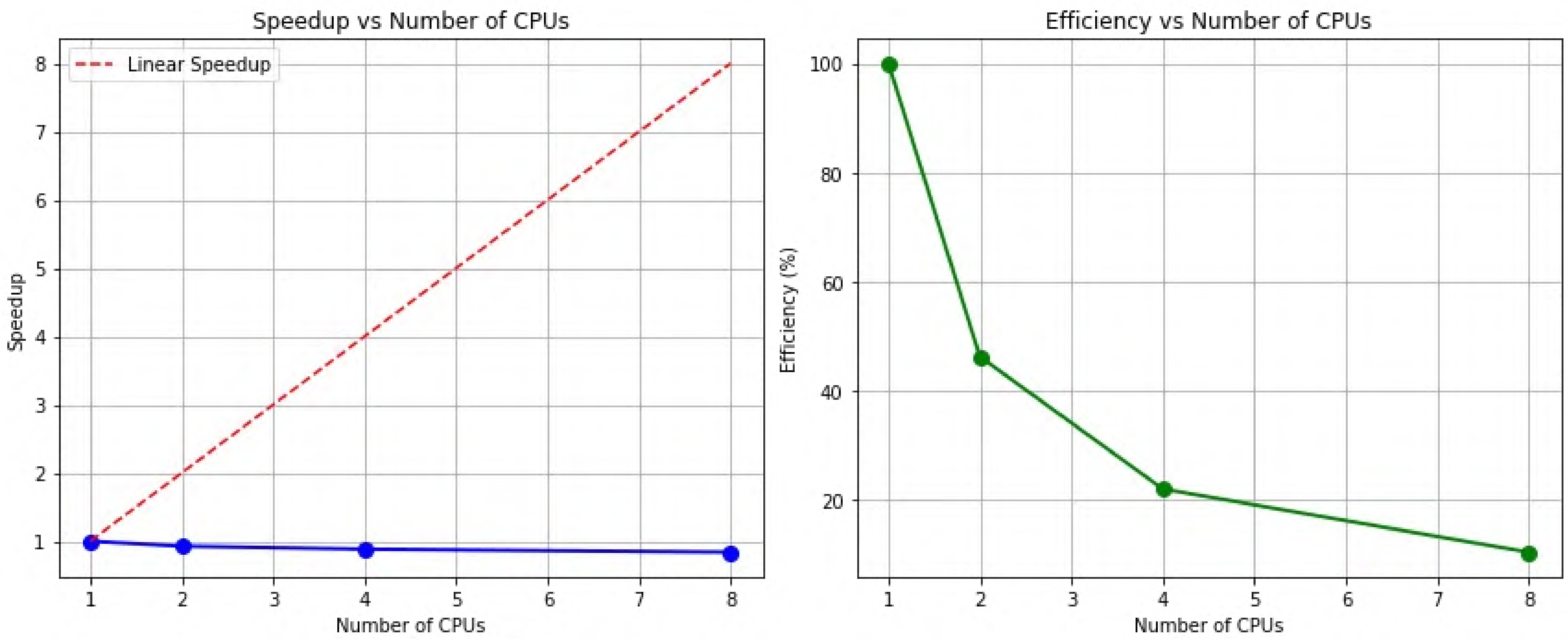


CLIP



Try Pitch

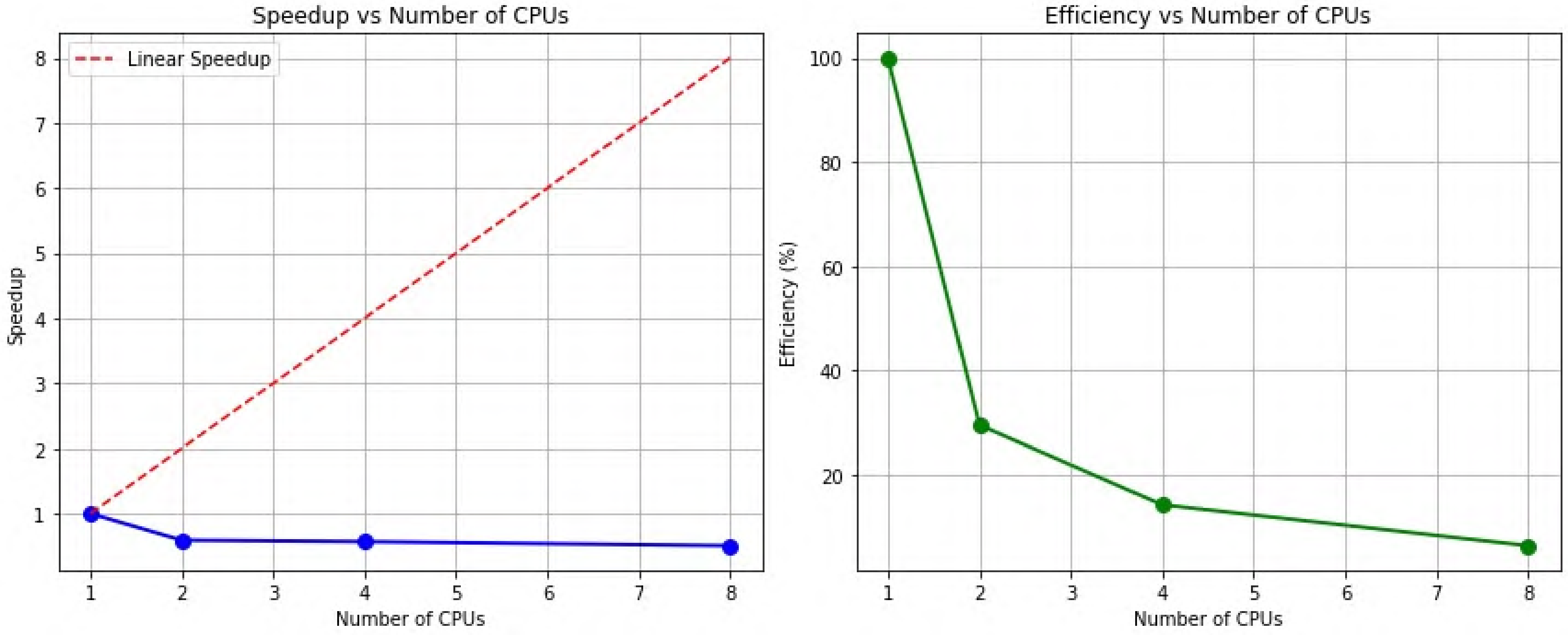
VAE



Parallel Processing Metrics:

CPUs	Time(s)	Speedup	Efficiency
1	299.00	1.00	100.00 %
2	323.00	0.93	46.28 %
4	339.00	0.88	22.05 %
8	357.00	0.84	10.47 %

CLIP

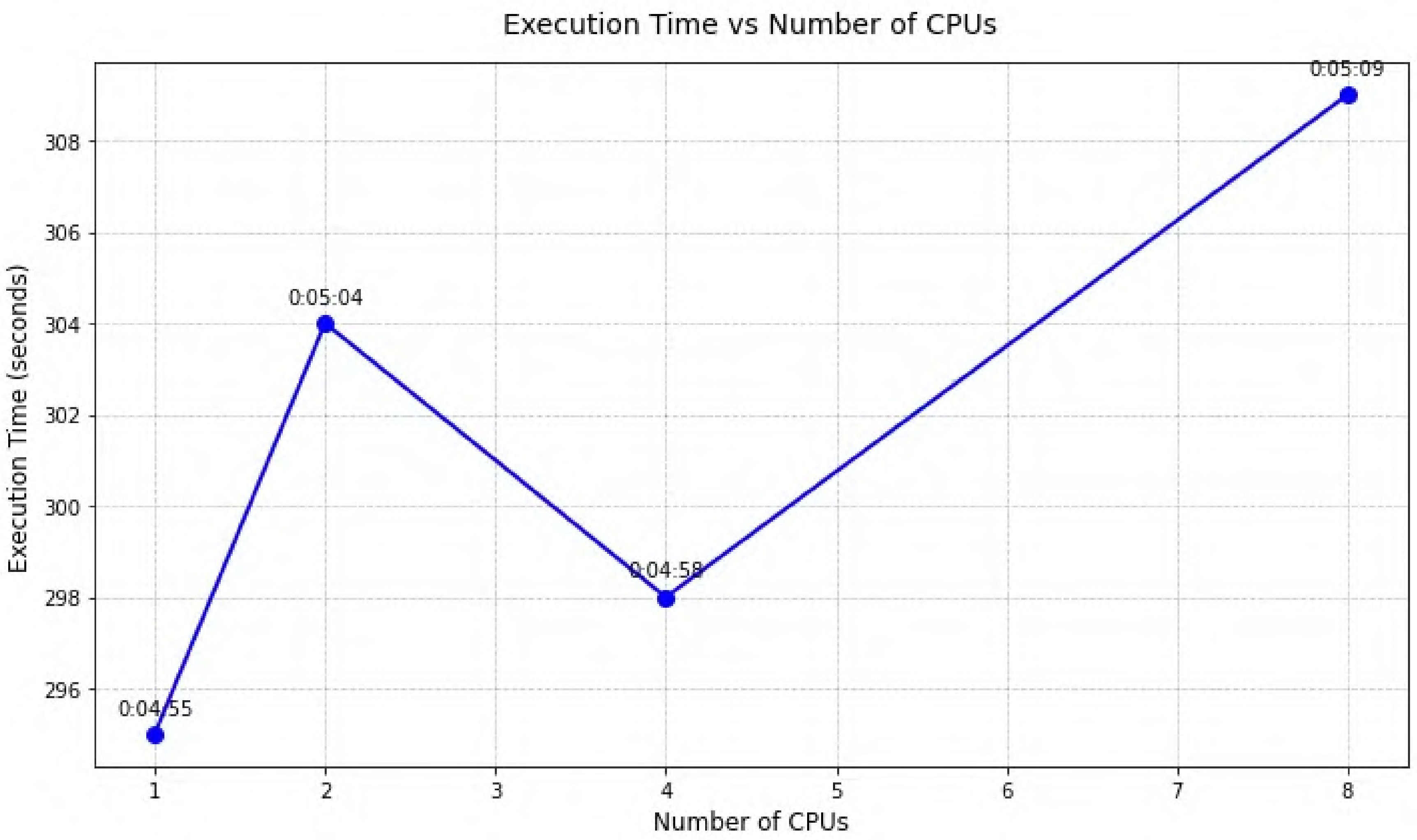


Parallel Processing Metrics:

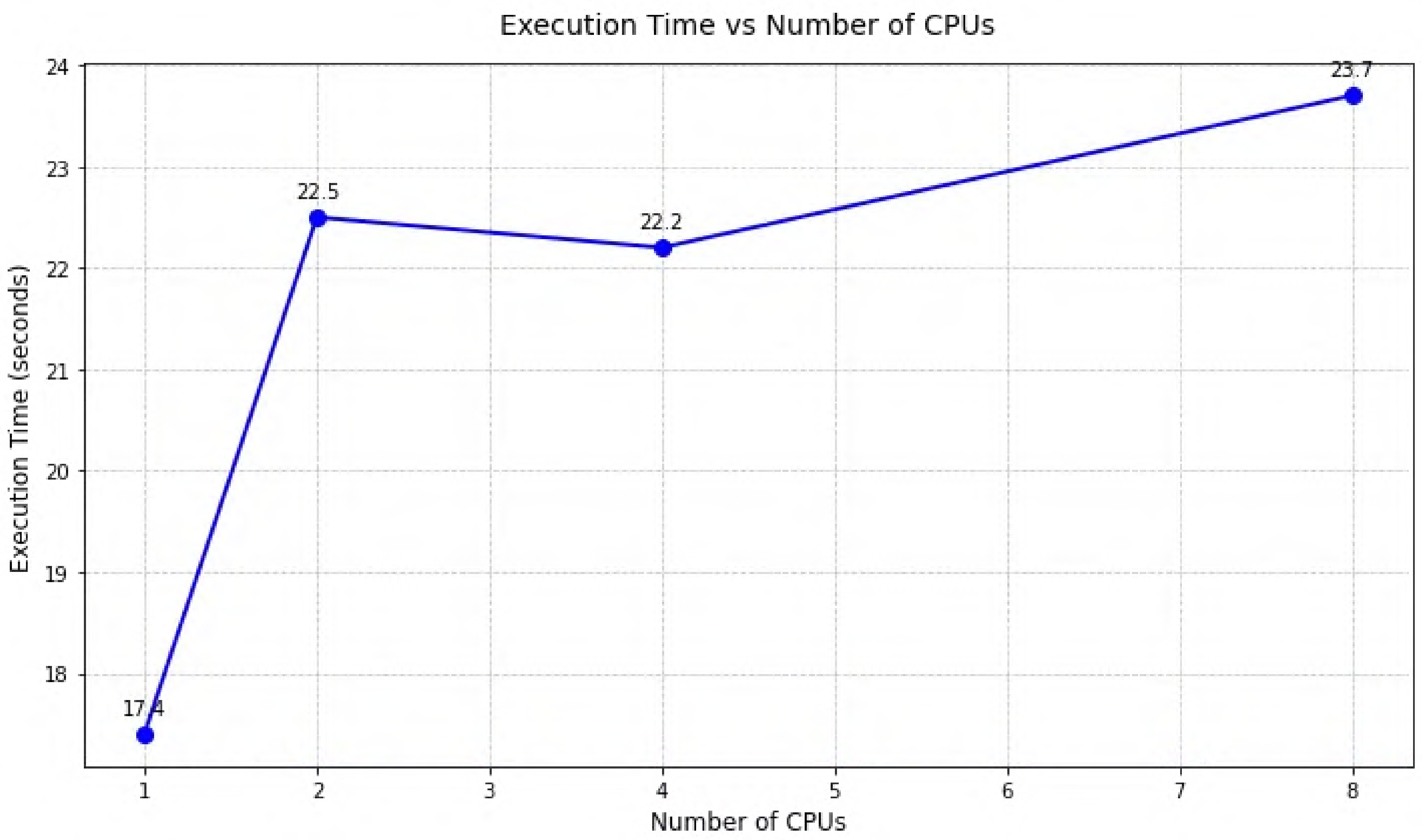
CPUs	Time(s)	Speedup	Efficiency
1	28.50	1.00	100.00 %
2	48.30	0.59	29.50 %
4	50.20	0.57	14.19 %
8	56.40	0.51	6.32 %

Joblib VAE and CLIP

VAE

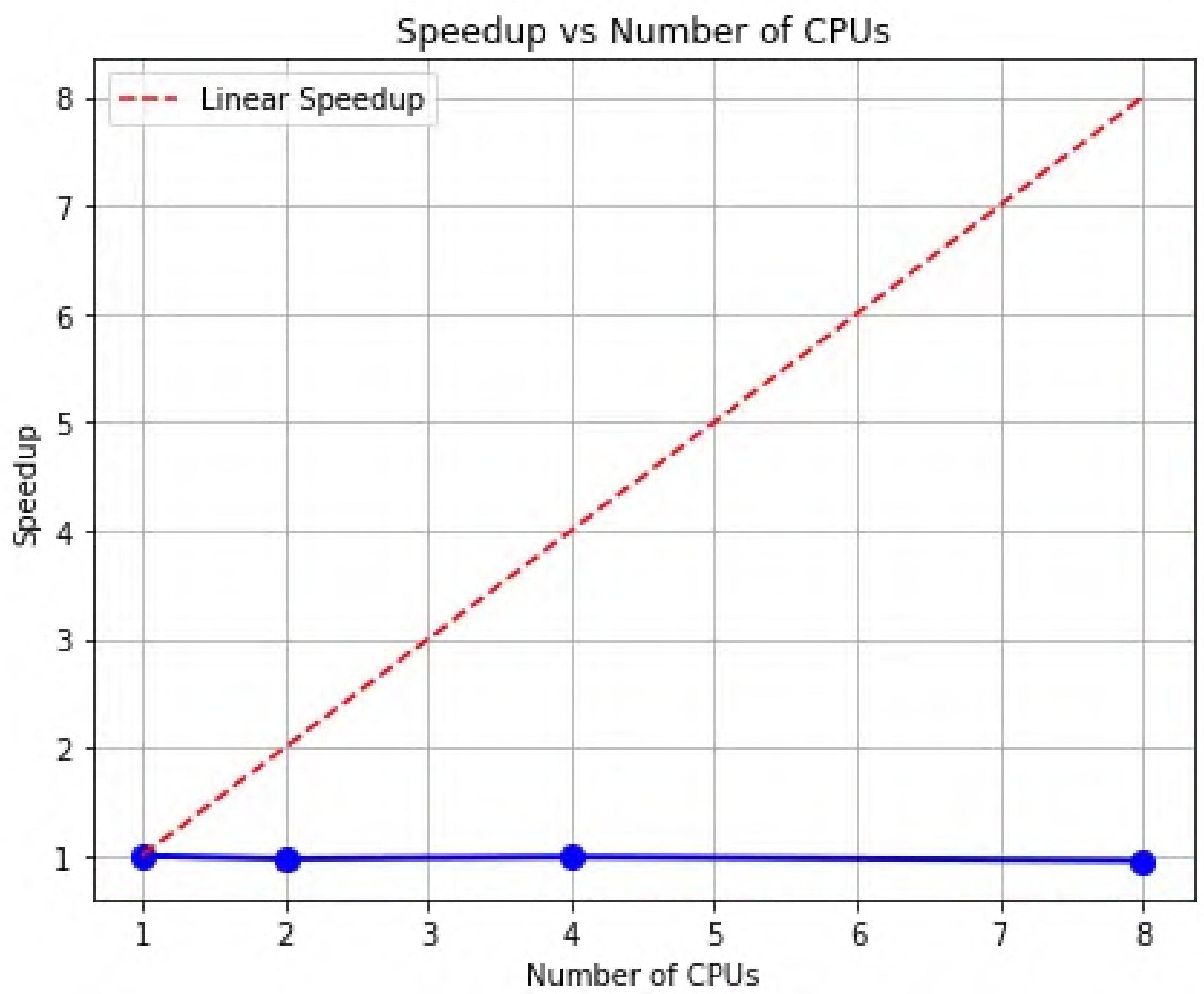


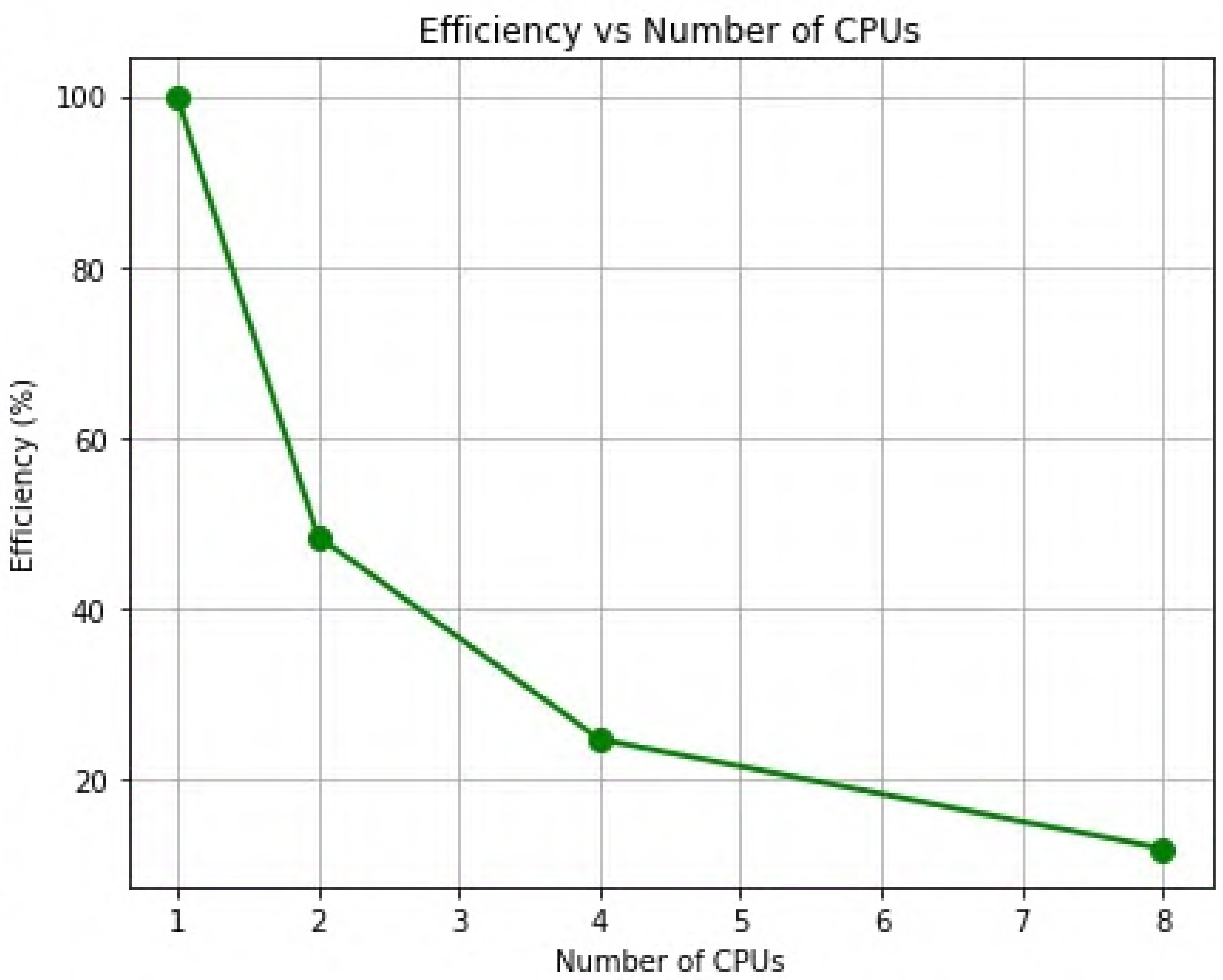
CLIP



Try Pitch

Joblib VAE



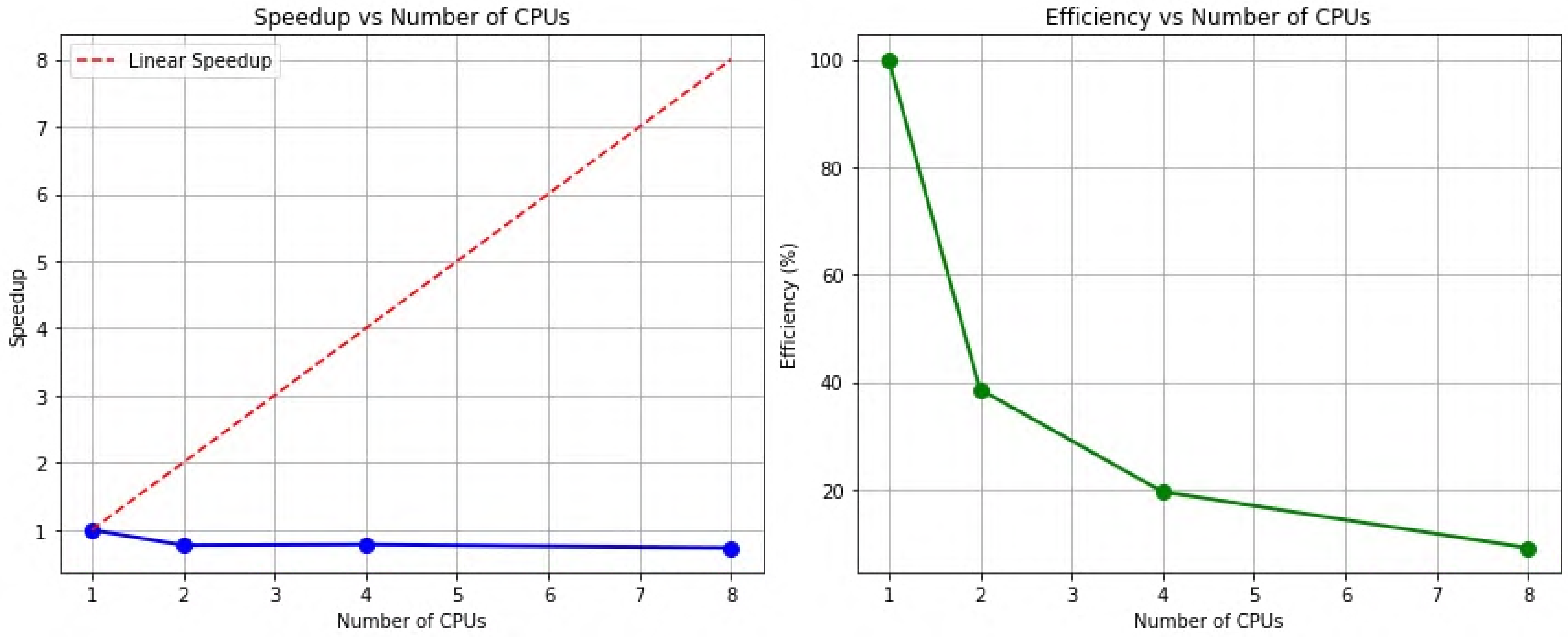


Parallel Processing Metrics:

CPUs	Time(s)	Speedup	Efficiency
1	295.00	1.00	100.00 %
2	304.00	0.97	48.52 %
4	298.00	0.99	24.75 %
8	309.00	0.95	11.93 %

Parallelizing across CPUs did not yield consistent speedups due to overheads and limited parallelization efficiencies.

JobLib CLIP

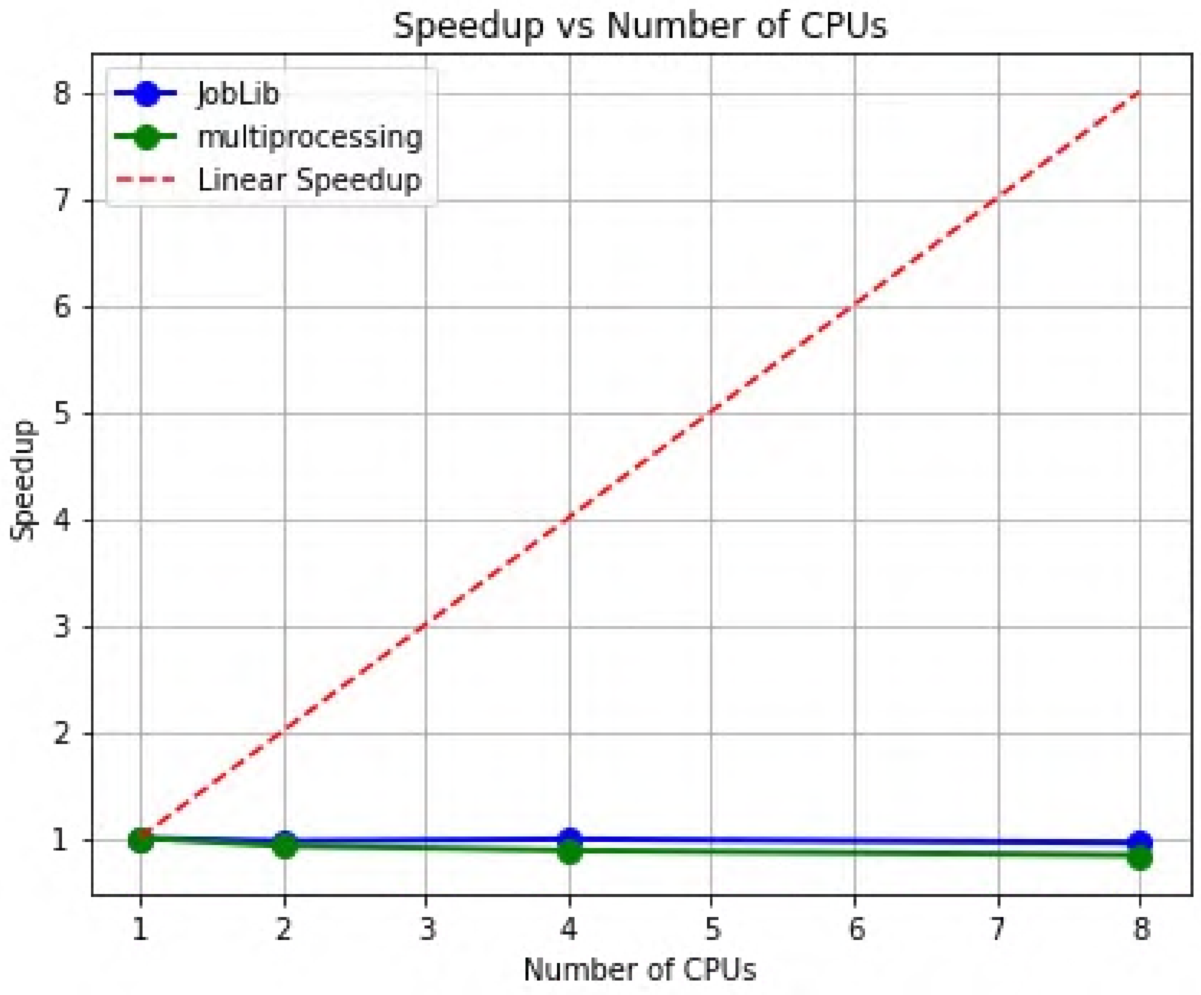


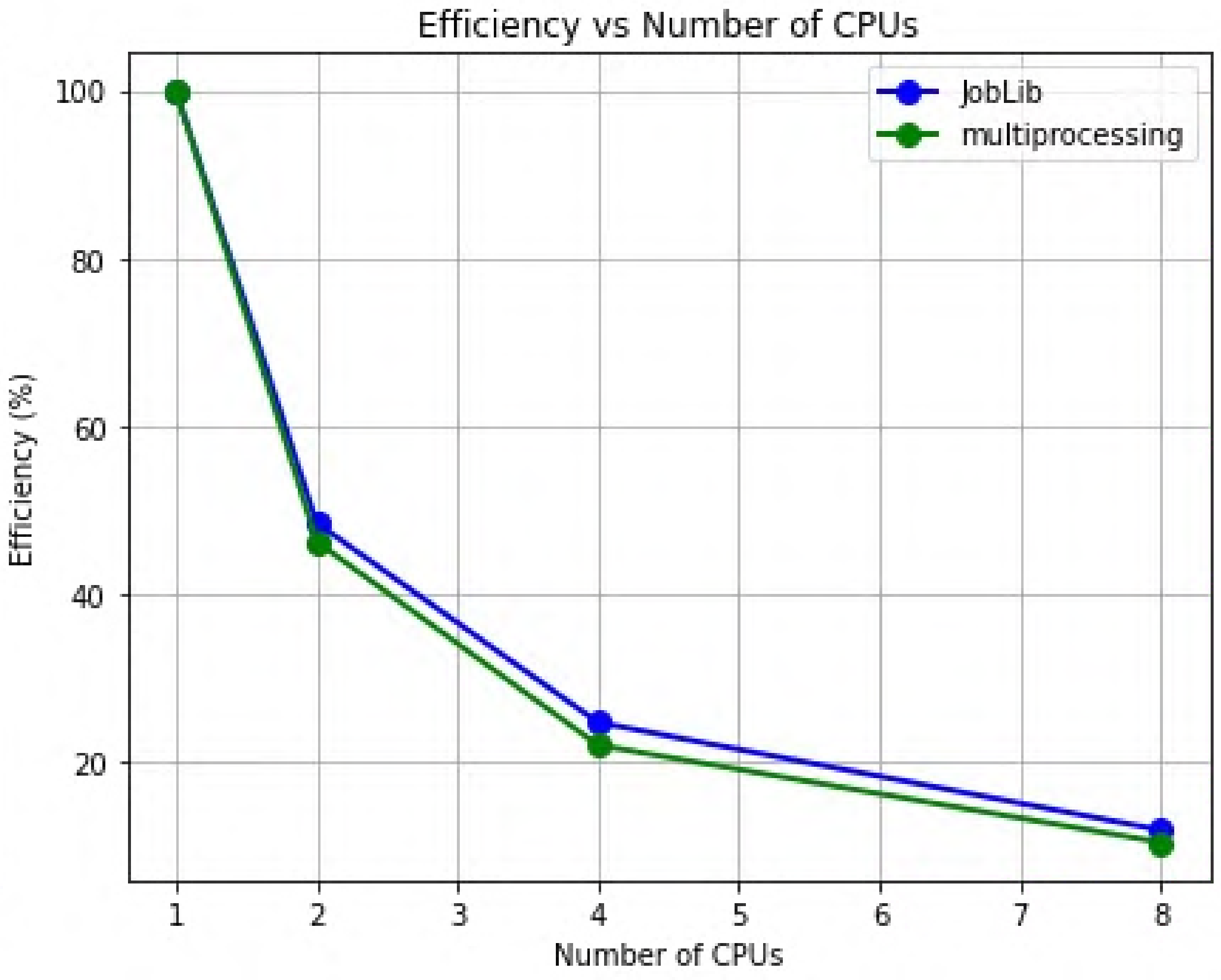
Parallel Processing Metrics:

CPUs	Time(s)	Speedup	Efficiency
1	17.40	1.00	100.00 %
2	22.50	0.77	38.67 %
4	22.20	0.78	19.59 %
8	23.70	0.73	9.18 %

Comparing joblib with multiprocessing

VAE



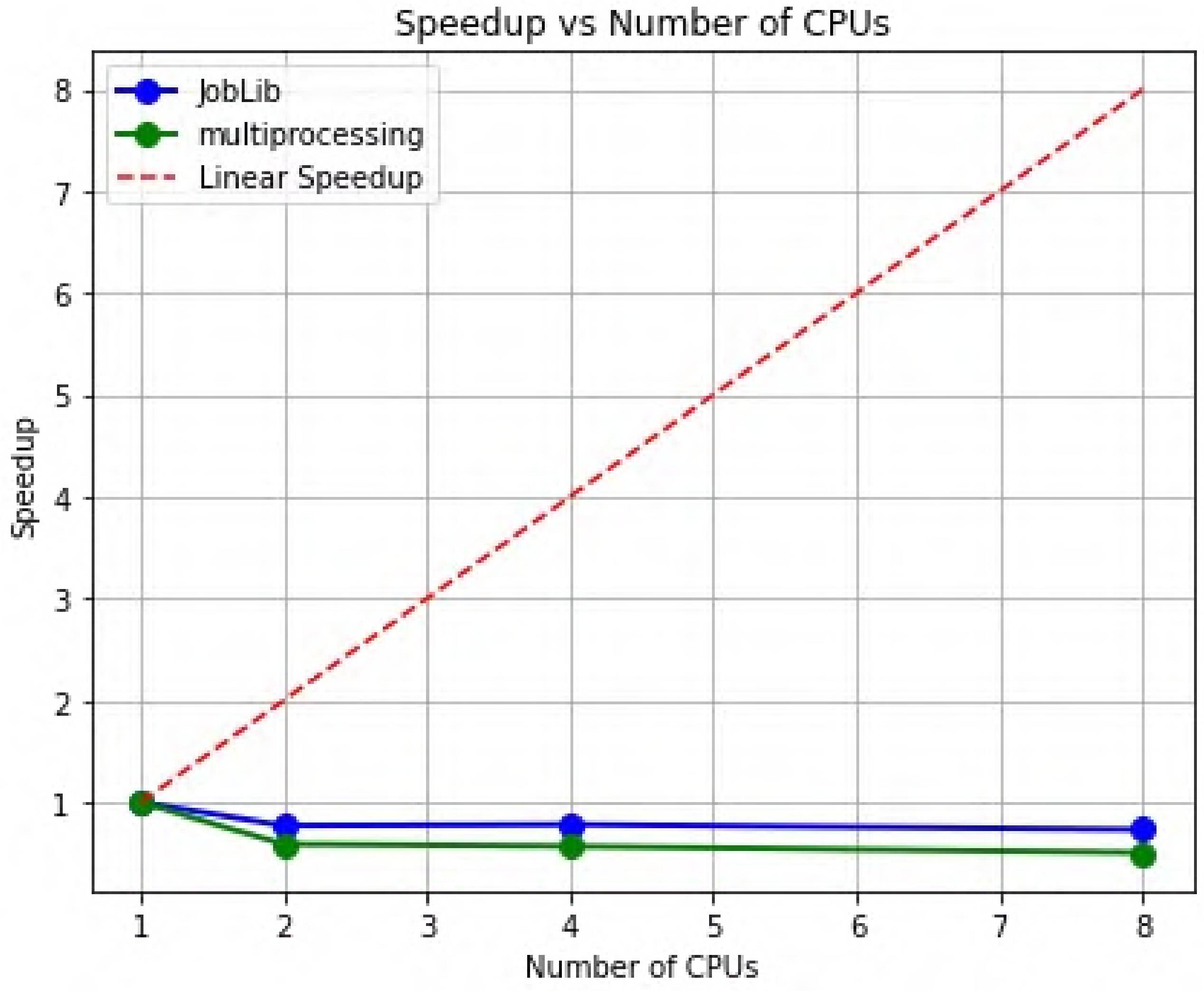


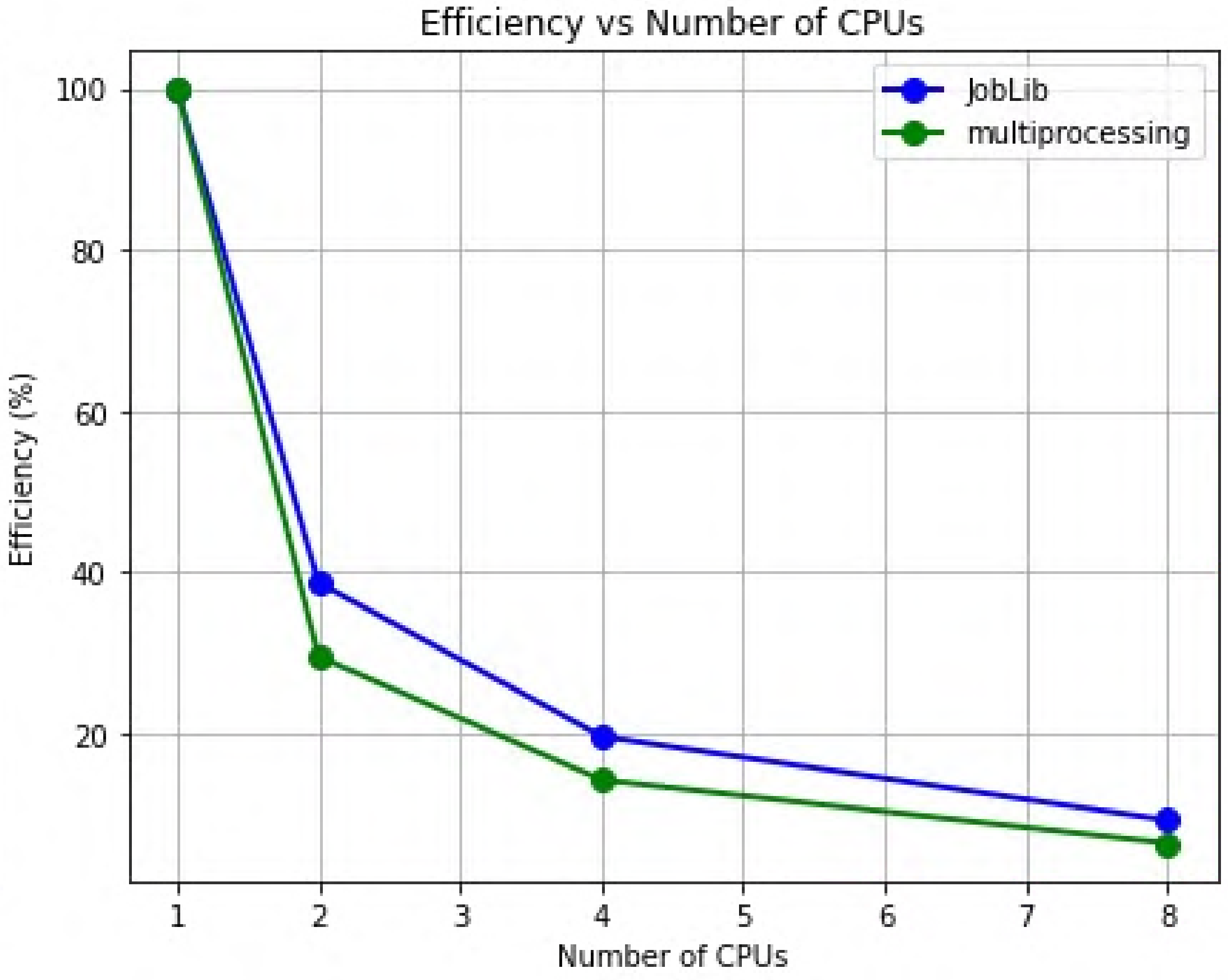
Parallel Processing Comparison:

CPUs	Time 1(s)	Time 2(s)	Speedup 1	Speedup 2	Eff 1(%)	Eff 2(%)
1	295.00	299.00	1.00	1.00	100.00	100.00
2	304.00	323.00	0.97	0.93	48.52	46.28
4	298.00	339.00	0.99	0.88	24.75	22.05
8	309.00	357.00	0.95	0.84	11.93	10.47

Comparing joblib with multiprocessing

CLIP

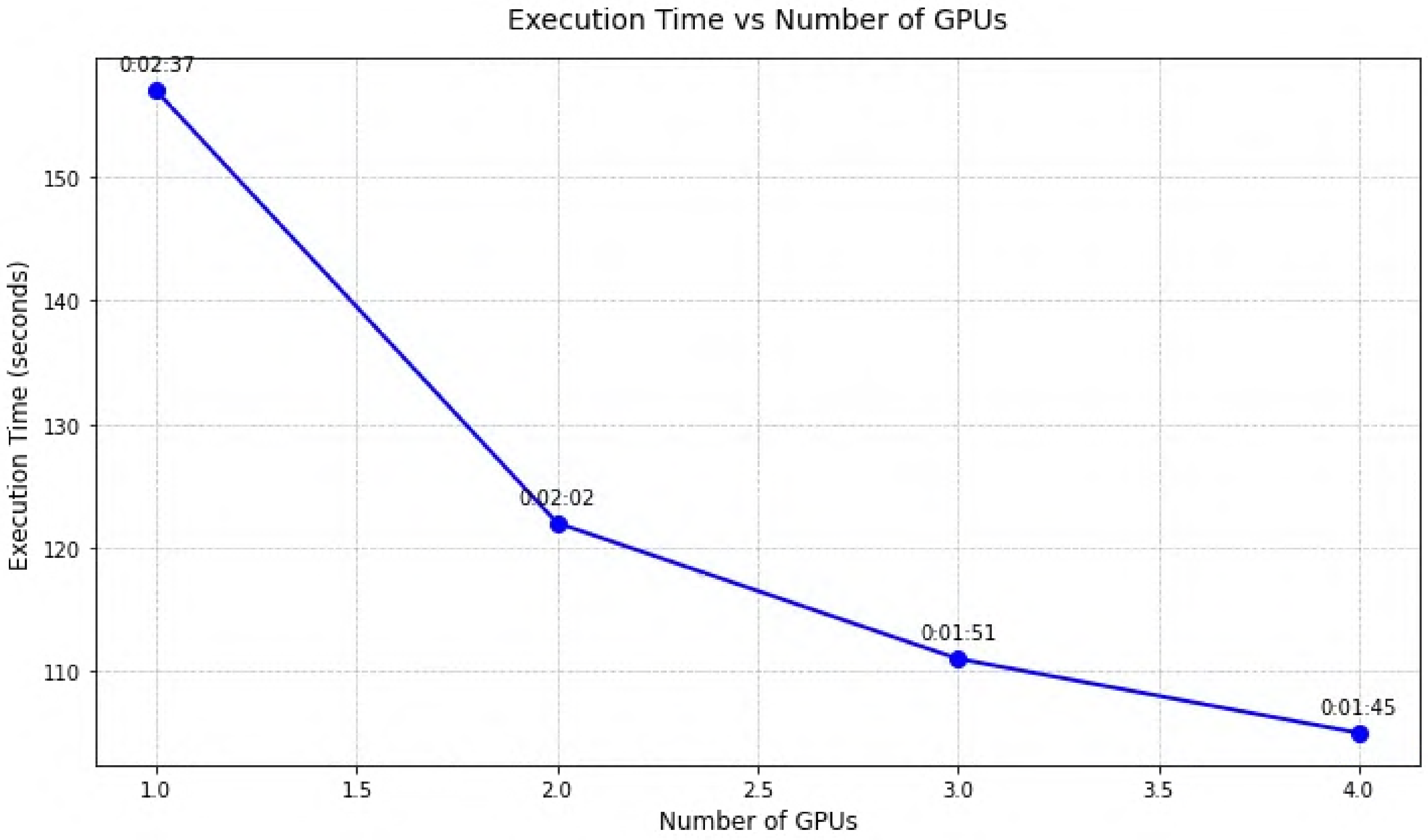





```
Parallel Processing Comparison:
CPUs    Time 1(s)   Time 2(s)   Speedup 1   Speedup 2   Eff 1(%)    Eff 2(%)
1       17.40       28.50       1.00        1.00        100.00      100.00
2       22.50       48.30       0.77        0.59        38.67       29.50
4       22.20       50.20       0.78        0.57        19.59       14.19
8       23.70       56.40       0.73        0.51        9.18        6.32
```

GPU Multiprocessing

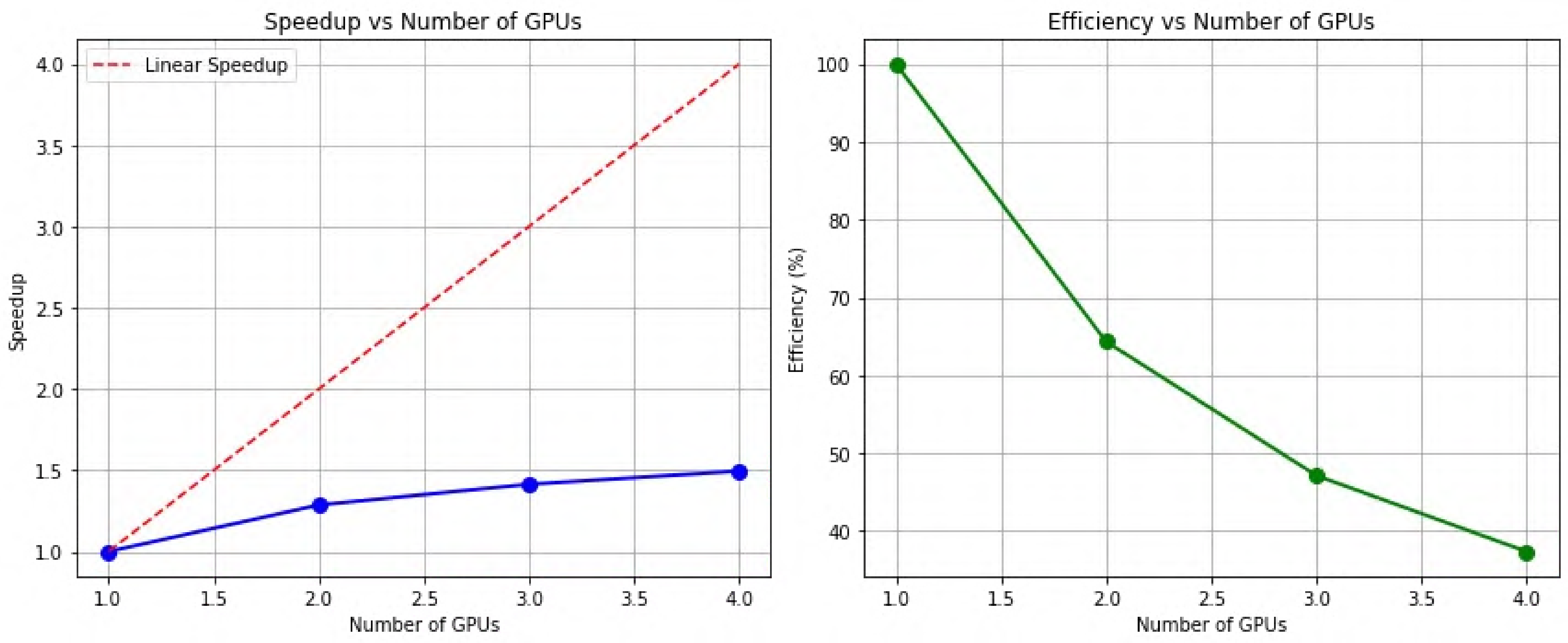
VAE



CLIP

Try Pitch

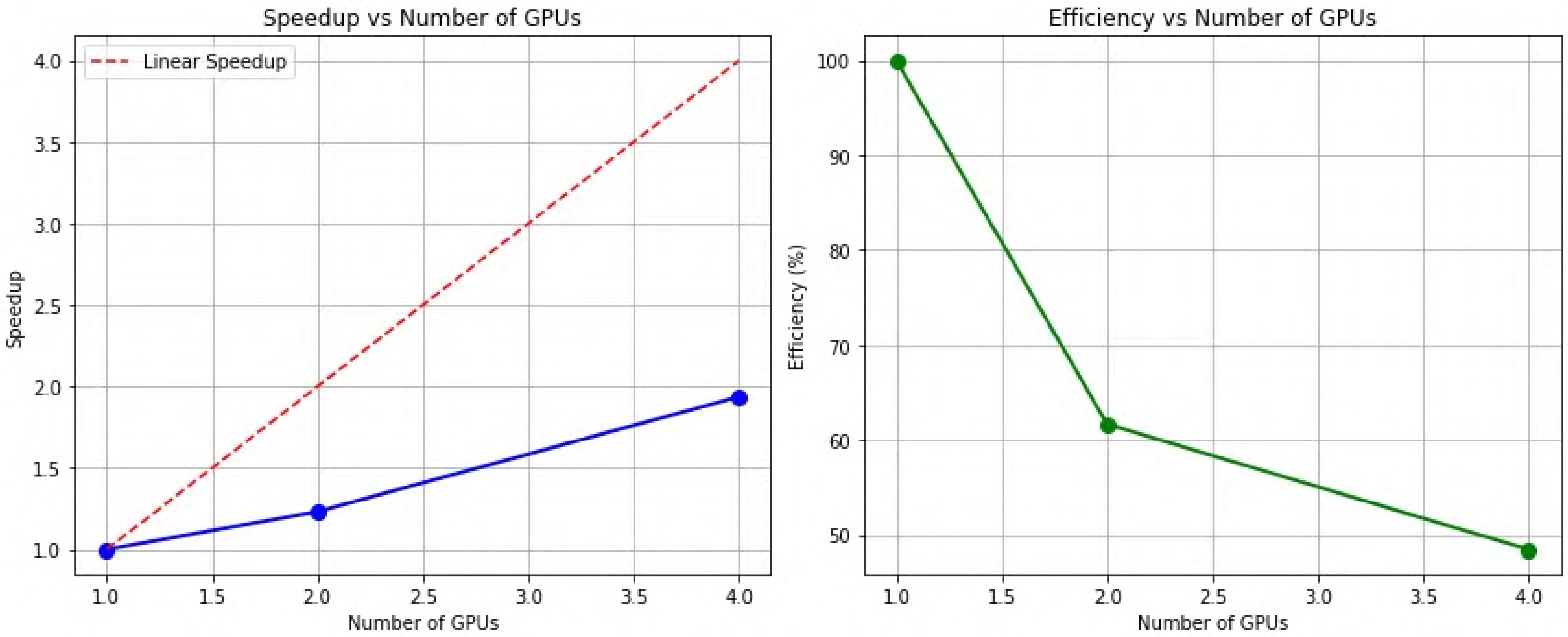
VAE



Parallel Processing Metrics:

CPUs	Time(s)	Speedup	Efficiency
1	157.00	1.00	100.00 %
2	122.00	1.29	64.34 %
3	111.00	1.41	47.15 %
4	105.00	1.50	37.38 %

CLIP



Parallel Processing Metrics:

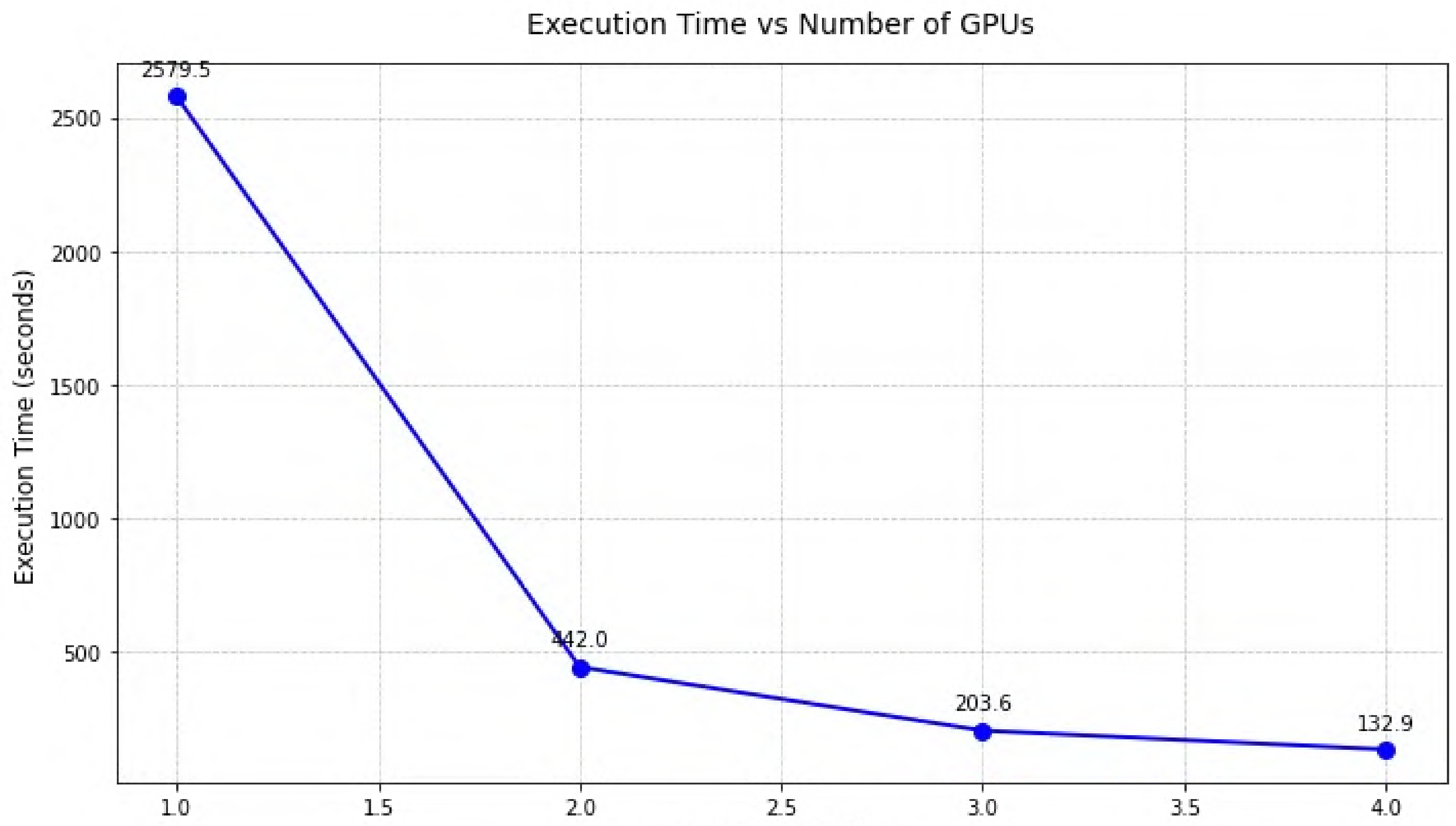
GPUs	Time(s)	Speedup	Efficiency
1	21.70	1.00	100.00 %
2	17.60	1.23	61.65 %
4	11.20	1.94	48.44 %

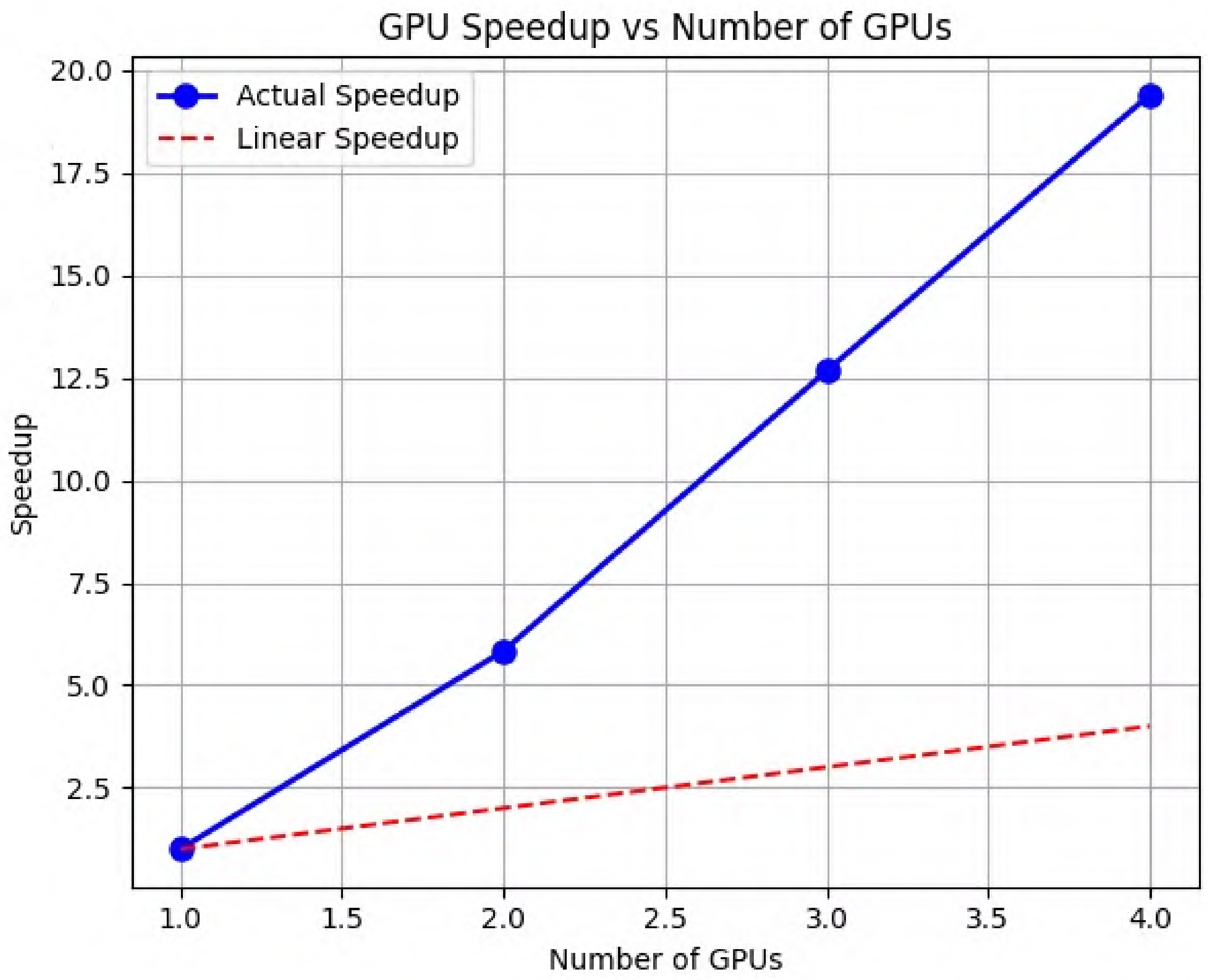
DDP Training used K80 GPUs

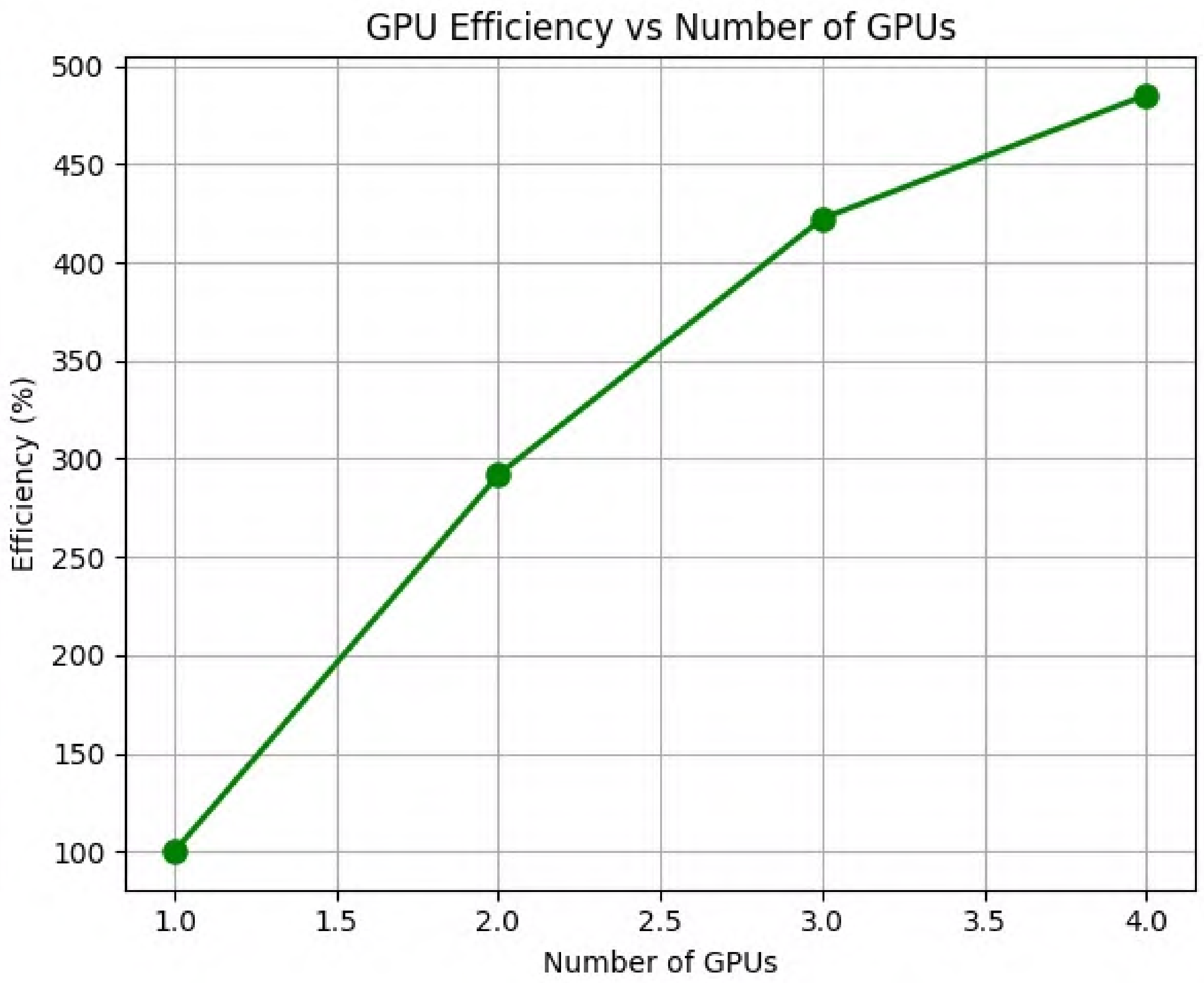
We used *mp.spawn*

which is multiprocessing due to

Python's threading is limited by GIL







Try Pitch

```
GPU Processing Metrics:
------------------------------------------------------------
GPUs   Time        Speedup    Efficiency
------------------------------------------------------------
1      2579.5      1.00       100.00    %
2      442.0       5.84       291.80    %
3      203.6       12.67      422.31    %
4      132.9       19.41      485.23    %

Times in seconds:
GPU 1: 2579.5 seconds
GPU 2: 442.0 seconds
GPU 3: 203.6 seconds
GPU 4: 132.9 seconds
```

06

Challenges faced

What are those challenges?

1

Disk quota exceeded

This is was the first and main problem for us since the dataset became 37.7GB after conversion of the pkl file

2

Daskification

Adding whole model inside cpus and converting into dask arrays and performing it made it to spill lot of memory.

cluster = LocalCluster(n_workers=n_workers, threads_per_worker=1, **memory_limit='8GB', memory_target_fraction=0.7, memory_spill_fraction=0.8, memory_pause_fraction=0.9**, processes=False)

3

Using scratch VAE

At first, I trained VAE to provide embedding for image but that wasn't efficient since we aren't using more dataset, the temporal features are less

07

Further Improvements

Try Pitch

Further Improvements

CUDA Kernels

Diverse Dataset

UI development

memory efficient framework

Real-Time Inference

Prompt suggestions

08

Conclusion

Our experiments reveal that while multiple CPUs can be employed for preprocessing stages (VAE and CLIP embeddings), their scalability and efficiency improvements are limited.

Parallelizing across CPUs did not yield consistent speedups **due to overheads and limited parallelization efficiencies**. Although **Joblib outperformed native multiprocessing** slightly, gains were modest.

In contrast, GPUs provided substantial speedups. Even though **a single GPU outperformed multiple CPUs**, scaling to multiple GPUs offered more pronounced improvements, especially during the training phase.
However, **the cost trade-off remains non-trivial: while GPU instances are more expensive**, their dramatically reduced processing times may justify their **cost in time-sensitive scenarios**.

For organizations or **researchers with limited GPU availability and abundant CPU** resources, a CPU-based pipeline could still be viable, particularly if cost savings outweigh longer processing times.

On the other hand, for high-throughput production environments, GPUs (possibly in combination with distributed frameworks) remain the superior choice, balancing speed, efficiency, and overall productivity.

9

References

Try Pitch

References

PyTorch Documentation
Hugging Face Transformers
MS-COCO
CLIP
Attention Mechanisms
Mixed Precision Training
Gradient Scaling
stabilityai/sd-vae-ft-mse VAE model
lucidrains/imagen-pytorch - Github for pytorch implemetation

Thank You
Any questions?

info@email.com