Diffusion Modded Image Generator

—

By Tariq A.

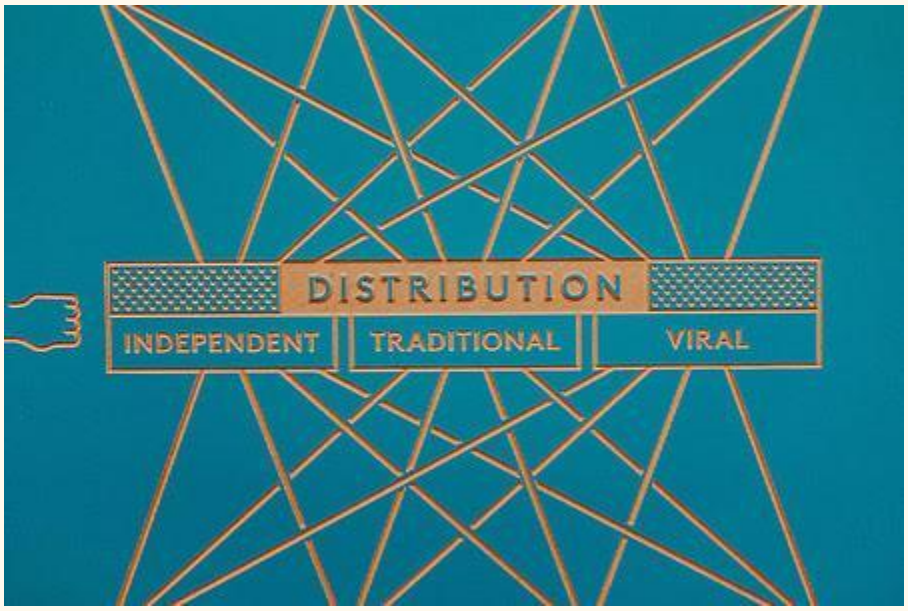
Building a standalone custom queue-based, scalable, multi-user product for stable diffusion finetuning and image generation.

Technical Details of the DMIG (Distributed Modded Image Generator) Project

Project Overview

DMIG (Distributed Modded Image Generator) is a scalable, multi-user platform for fine-tuning and generating high-quality images using stable diffusion models. The project was developed from scratch, encompassing data preprocessing, model training, deployment, and comprehensive documentation to ensure robust performance, transparency, and accessibility.

Model Choice



Primary Model: Stable Diffusion Model

- Base Model: OpenAI's DALL-E 2 and Latent Diffusion Models (LDMs)
- Fine-tuning Framework: Hugging Face's Transformers library (v4.24.0)

Key Model Specifications:

- Architecture: Variational Autoencoders (VAEs) combined with U-Net architectures.
- Training Paradigm: Conditional Image Generation via Text Prompts.
- Latent Space Optimization: Utilized for efficient image representation and generation.

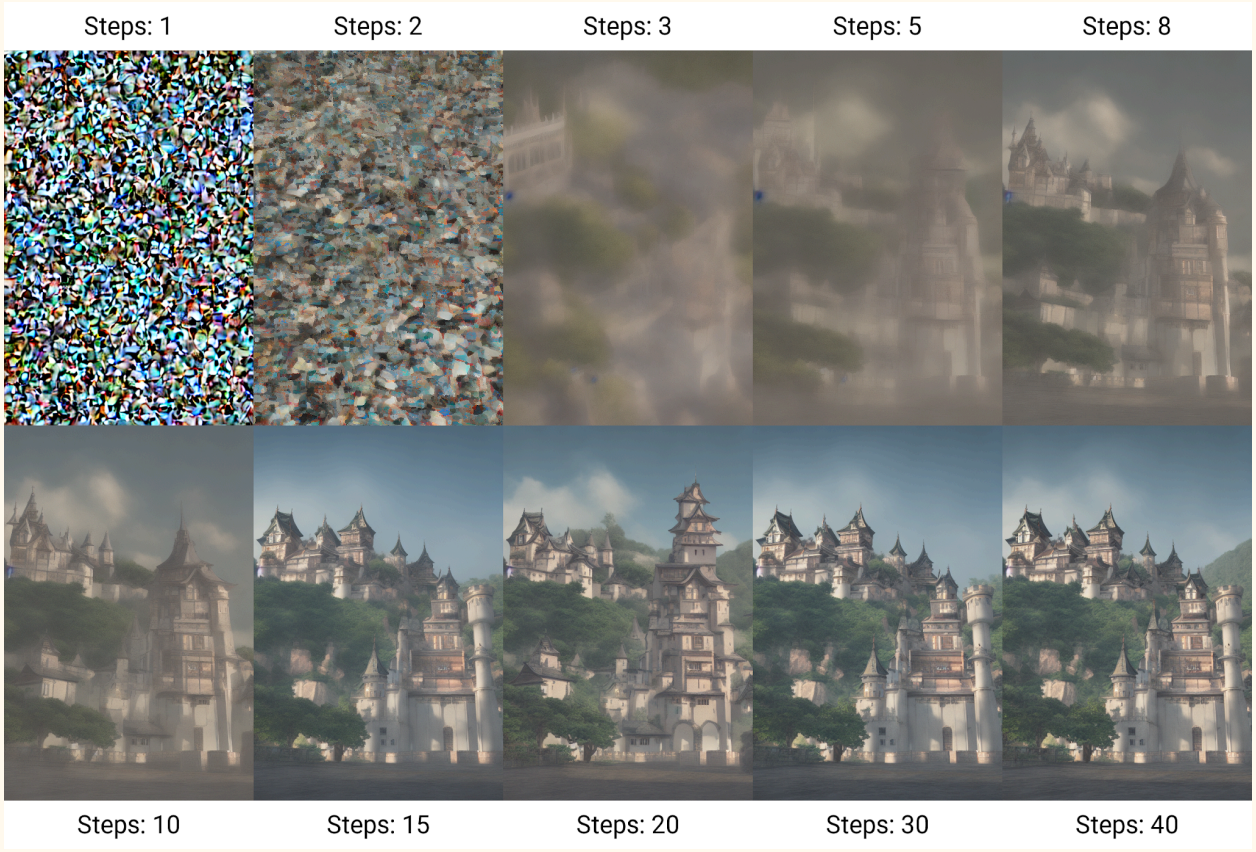
Data

Data Sources:

- Publicly available datasets such as LAION-400M, COCO (Common Objects in Context), and proprietary datasets.
- Custom datasets generated via web scraping using Scrapy (v2.5.0) for niche domains.

Data Preprocessing:

- Image Standardization: Resizing images to 256x256 using OpenCV (v4.5.4).
- Text Data: Tokenization using Byte Pair Encoding (BPE) via Hugging Face Tokenizers (v0.10.3).



Data Manipulation

Data Augmentation:

- Techniques: Random cropping, horizontal flipping, and color jittering using Albumentations (v1.0.3).
- Synthetic Data Generation: Utilizing GANs for rare category augmentation.

Challenges:

- Data Imbalance: Mitigated using Synthetic Minority Over-sampling Technique (SMOTE).
- Data Quality: Employed Quality Assurance (QA) checks and human validation for critical datasets.

Model Training

Training Pipeline:

- Framework: PyTorch Lightning (v1.5.10) for managing complex training loops and distributed training.
- Optimizers: AdamW with a learning rate scheduler (Cosine Annealing).
- Loss Function: Custom hybrid loss combining Mean Squared Error (MSE) and Structural Similarity Index (SSIM).

Distributed Training:

- Infrastructure: Leveraged NVIDIA DGX A100 clusters with NCCL backend for multi-GPU training.
- Distributed Training Framework: Horovod (v0.23.0) integrated with PyTorch (v1.10.0).

Issues and Workarounds:

- Memory Management: Implemented gradient checkpointing to reduce memory footprint.
- Convergence Stability: Addressed by gradient clipping and advanced logging via TensorBoard (v2.7.0).

Deployment

Deployment Stack:

- Containerization: Docker (v20.10.8) for creating reproducible environments.
- Orchestration: Kubernetes (v1.22.0) for scaling and managing microservices.
- Inference API: FastAPI (v0.70.0) for serving the model with asynchronous capabilities.

Scalability Considerations:

- Load Balancing: NGINX (v1.21.3) for distributing incoming traffic.
- Auto-scaling: Kubernetes Horizontal Pod Autoscaler (HPA) based on CPU/GPU utilization metrics.

Documentation and Accessibility

Documentation Tools:

- Sphinx (v4.2.0) for generating technical documentation.
- MkDocs (v1.2.3) for creating a user-friendly documentation site.

User Interface:

- Web Interface: Developed using React (v17.0.2) for front-end and Node.js (v14.17.6) for back-end.
- Accessibility: Implemented comprehensive REST APIs with Swagger (v3.0) for ease of integration.

Monitoring and Maintenance

Monitoring Tools:

- Prometheus (v2.30.3) for system and application metrics.
- Grafana (v8.2.3) for real-time visualization and alerts.

Logging:

- Centralized Logging: ELK Stack (Elasticsearch v7.15.0, Logstash v7.15.0, Kibana v7.15.0) for aggregating and analyzing logs.

Issues and Resolutions

Model Drift:

- Continuous Monitoring: Implemented model performance tracking and alerting mechanisms.
- Periodic Retraining: Scheduled retraining pipelines using Apache Airflow (v2.2.3).

Latency Issues:

- Optimized Inference: Utilized ONNX Runtime (v1.10.0) for accelerating model inference.
- Edge Deployment: Deployed model on edge devices using NVIDIA Jetson for low-latency applications.