Text-to-Image Generation System using Stable Diffusion

Author: Neha Dharanu
Date: December 2025

Table of Contents

---

1. Introduction

Generating images from natural language descriptions is one of the most visible and impactful applications of Generative AI. Text-to-image systems enable creative design, education, rapid prototyping, product visualization, and accessibility applications. Recent progress is largely driven by diffusion models, multimodal representation learning, and open-source model checkpoints.

This project implements a complete text-to-image generation system using **Stable Diffusion** with **CLIP-based text conditioning**, including:

- A modular inference pipeline
- Configurable generation controls
- An interactive web interface for demonstration
- An evaluation workflow using standard metrics and structured experiments

At a high level, text is embedded using a transformer-based encoder, latent noise is iteratively denoised using a UNet with cross-attention to align with prompt semantics, and the final latent representation is decoded into an RGB image using a VAE decoder. The project includes engineering considerations such as caching, inference-time constraints, reproducibility, and deployment stability.

---

2. Problem Statement

Traditional GAN-based generators can produce sharp images but often face training stability issues and reduced controllability, especially for open-ended natural language prompts. Direct diffusion in pixel space is computationally expensive, and interactive deployments require careful optimization.

This project addresses:
How can we build a text-to-image system that produces visually coherent and semantically aligned images from user prompts, while remaining computationally feasible, configurable, and deployable through a user-friendly interface?

Key subproblems:

1. Text understanding and conditioning (prompt embeddings)
2. Efficient generation (latent diffusion)

3. Prompt alignment and controllability (CFG, steps, scheduler choices)
4. Evaluation (FID, Inception Score, qualitative comparisons)
5. Deployment usability (web demo and clean UI)

---

3. Background and Related Concepts

3.1 Text-to-Image Generation

Text-to-image generation aims to produce images consistent with prompt semantics including objects, attributes, lighting, style, and spatial composition. Modern approaches use conditioning via attention mechanisms that let the generator focus on different prompt components throughout generation.

3.2 Diffusion Models

Diffusion models learn to reverse a gradual noising process. A denoising network predicts and removes noise iteratively, producing high-quality images with stable training characteristics.

3.3 Latent Diffusion Models

Pixel-space diffusion is expensive. Latent diffusion performs denoising in a compressed latent space (example: 64×64×4) mapped to and from pixel space using a VAE. This reduces compute and memory without major quality loss.

3.4 CLIP and Text Conditioning

CLIP aligns text and images in a joint representation space. In Stable Diffusion, the CLIP text encoder produces embeddings which condition the UNet via cross-attention to align generated visual features with prompt semantics.

3.5 Stable Diffusion Overview

Stable Diffusion combines:

- CLIP text encoder
- UNet latent denoiser
- VAE decoder
- Noise scheduler (DDIM, PNDM, Euler, etc.)
- Classifier-Free Guidance (CFG) for prompt adherence

---

4. Proposed Solution

This project implements a modular stable diffusion pipeline with:

- Local fine-tuned checkpoint loading
- Prompt-to-image generation via CLIP-conditioned latent diffusion

- Configurable inference controls (CFG scale, inference steps, scheduler selection during experiments)
- Evaluation pipeline (FID, Inception Score)
- Streamlit web application providing an interactive demo and output download support

---

5. System Architecture

5.1 High-Level Data Flow

1. User enters prompt
2. Prompt is tokenized (fixed max length)
3. CLIP text encoder produces embeddings
4. Random latent noise is initialized
5. UNet denoises over T steps
6. Cross-attention injects text conditioning
7. Final latent decoded using VAE
8. Output image produced at 512×512 RGB
9. Evaluation performed (offline / batch)
10. Web UI displays results and allows download

5.2 Component-Level Architecture

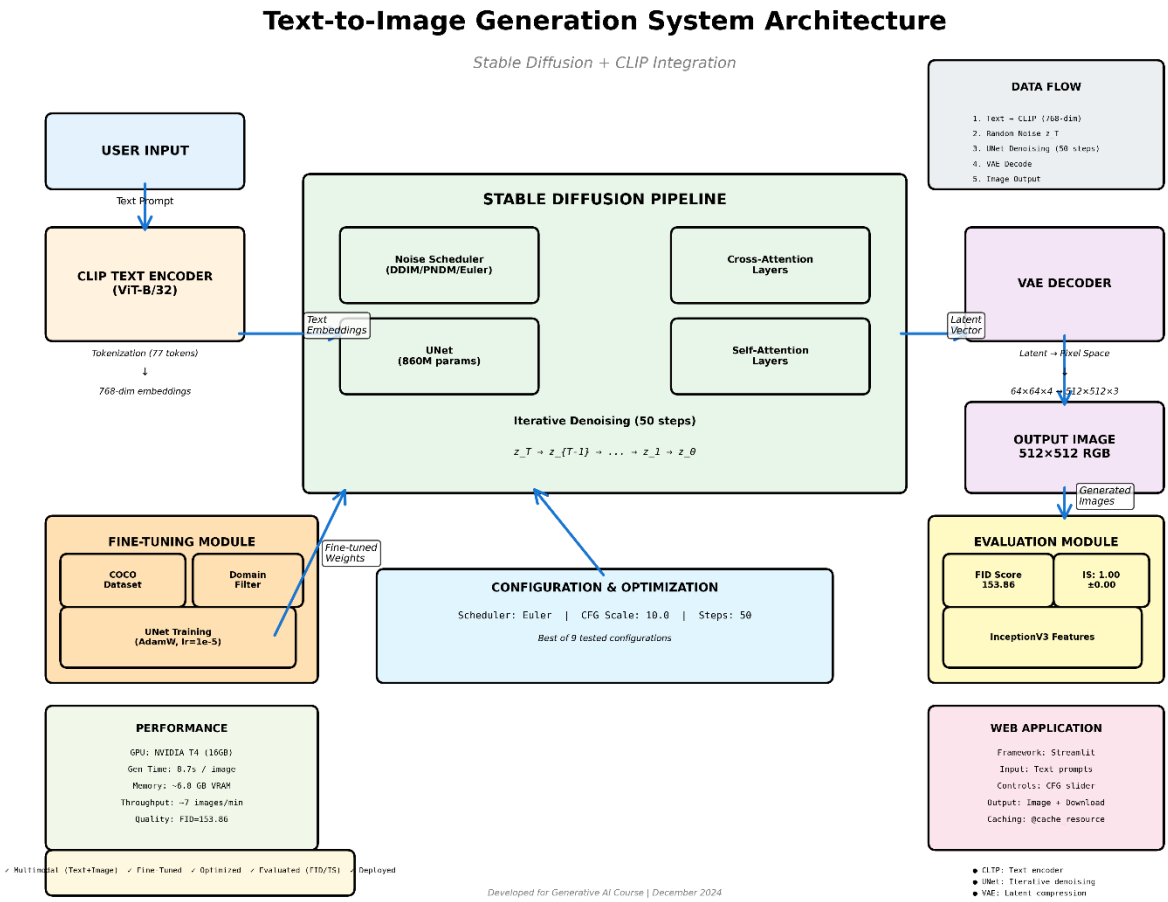

Figure 1: Text-to-Image Generation System Architecture (Stable Diffusion + CLIP Integration)

Modules:

- User Input
- CLIP Text Encoder
- Stable Diffusion Pipeline (UNet + Scheduler + Attention)
- VAE Decoder
- Output Renderer
- Evaluation Module
- Configuration and Optimization Module
- Web Application Module

5.3 CLIP Text Encoder Module

Purpose: convert prompt into contextual token embeddings.
Steps: tokenization, padding/truncation, embedding lookup, transformer encoding.
Output: embedding sequence (typically 768-d per token).

5.4 Stable Diffusion Pipeline Module

Orchestrates inference with prompt embeddings, latents, scheduler rules, CFG, and step count. Outputs final denoised latents.

5.5 UNet Denoising Network

Predicts noise at each timestep. Uses multi-scale blocks and skip connections, and conditioning via cross-attention.

5.6 Attention Mechanisms

- Self-attention: relationships within latent features
- Cross-attention: aligns latent features with text embeddings
  Cross-attention is critical for prompt alignment.

5.7 Noise Scheduler and Sampling

Schedulers define timestep trajectories and update rules. This project evaluates multiple schedulers in experiments, including:

- DDIM
- PNDM
- Euler
  Schedulers affect speed, sharpness, and stability.

5.8 Latent Space Representation

Denoising occurs on compressed latents (example 64×64×4) rather than 512×512×3 pixels, reducing compute.

5.9 VAE Decoder Module

Converts final latent tensor back to RGB image. Decoder quality impacts sharpness and color fidelity.

5.10 Evaluation Module

Provides measurable quality indicators:

- **FID** compares feature distributions vs real images
- **Inception Score** reflects clarity and diversity via classifier confidence
  Evaluation is performed via batch scripts, not inside the live demo.

5.11 Web Application Module

A Streamlit interface provides:

- Prompt input
- Generate action
- Display generated image
- Download output
  The UI is designed to be minimal and demo-focused.

5.12 Configuration and Optimization Module

Tracks scheduler, CFG scale, step count, resolution, best configurations. Supports reproducibility and experiment reporting.

---

6. Implementation Details

6.1 Technology Stack and Libraries

- Python
- PyTorch
- Hugging Face Diffusers
- Transformers (CLIP)
- Streamlit
- PIL / OpenCV
- NumPy
  Hardware (example): NVIDIA T4 GPU (16GB) for evaluation runs.

6.2 Pipeline Construction and Model Loading

- Loads Stable Diffusion pipeline and local fine-tuned checkpoint
- Uses caching to avoid reloading per interaction
- Ensures stable inference runtime

6.3 Prompt Handling and Tokenization

- Validates input prompt
- Tokenizes with CLIP tokenizer
- Embeds prompt via text encoder
  Future enhancement: negative prompts in the UI for better control.

6.4 Inference Parameters and Controls

Key parameters used in experiments:

- CFG scale: prompt adherence tradeoff
- Inference steps: quality vs latency tradeoff
- Scheduler: sampling style and stability

6.5 End-to-End Generation Flow

1. Read prompt
2. Encode prompt to embeddings
3. Initialize latents
4. For each timestep: UNet predicts noise and scheduler updates latents

5. Decode final latent with VAE
6. Post-process and display/save outputs
7. Evaluation runs offline on generated outputs

6.6 Web App Design and User Experience

Design goals: clean, minimal, prompt-first flow.
User workflow:

1. Open app
2. Enter prompt
3. Generate image
4. View image
5. Download image

6.6.1 Web Application Screenshots

The deployed Streamlit application demonstrates the complete user workflow from prompt input to image generation and download.

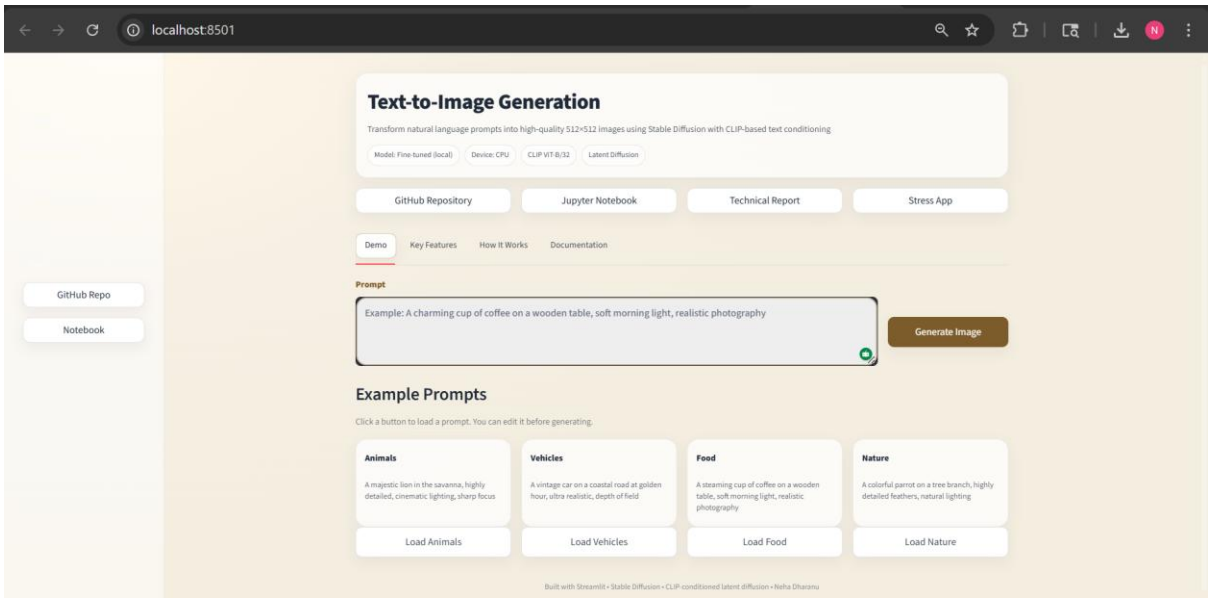

Figure 5 : Application Homepage
Main interface featuring prompt input area, example prompt gallery with four categories (Animals, Vehicles, Food, Nature), and the generate button. The beige color scheme provides a professional, clean aesthetic.

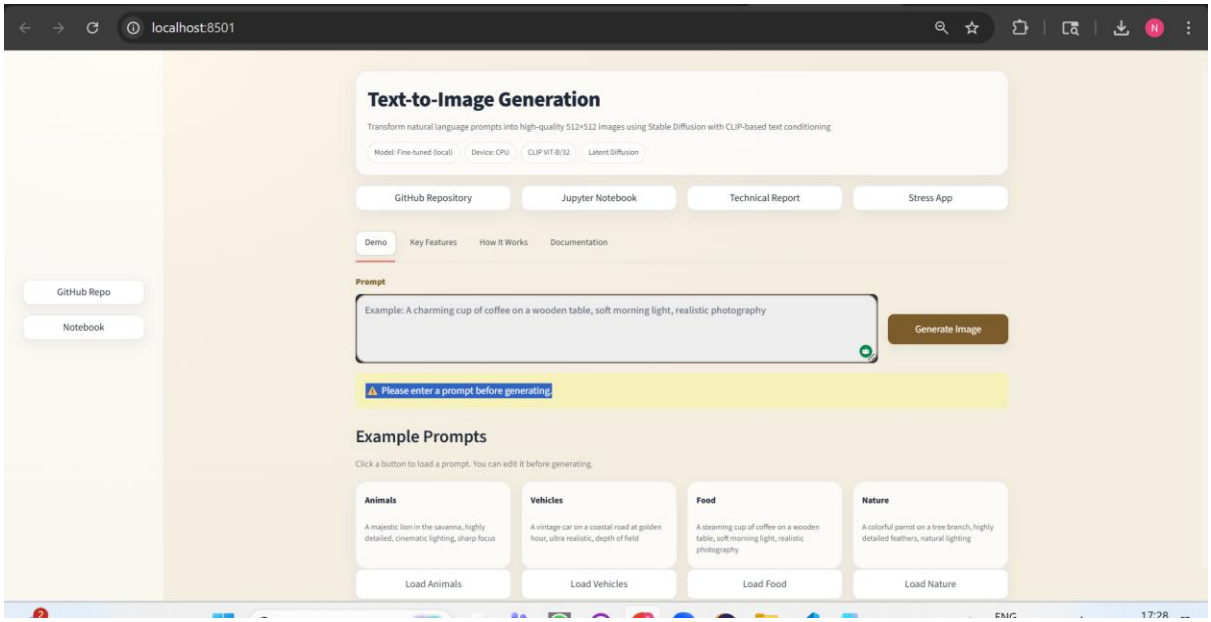

Figure 6: Input field validation

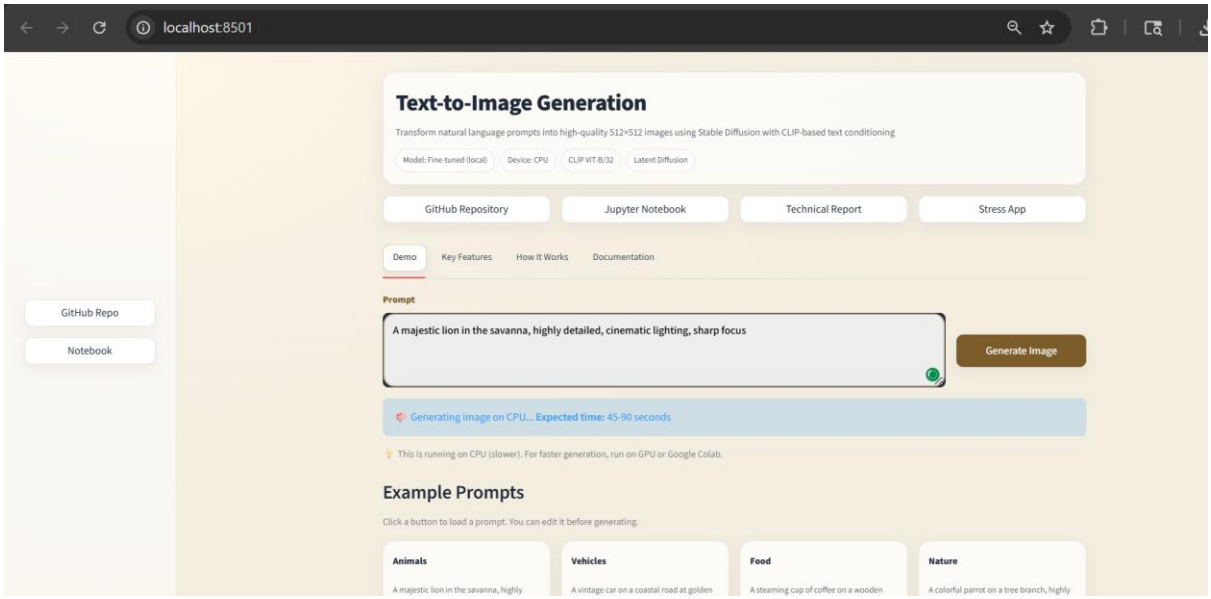

Figure 7: Generation in Progress

Status indicator showing expected generation time (45-90 seconds on CPU, 8-15 seconds on GPU) with helpful user guidance.

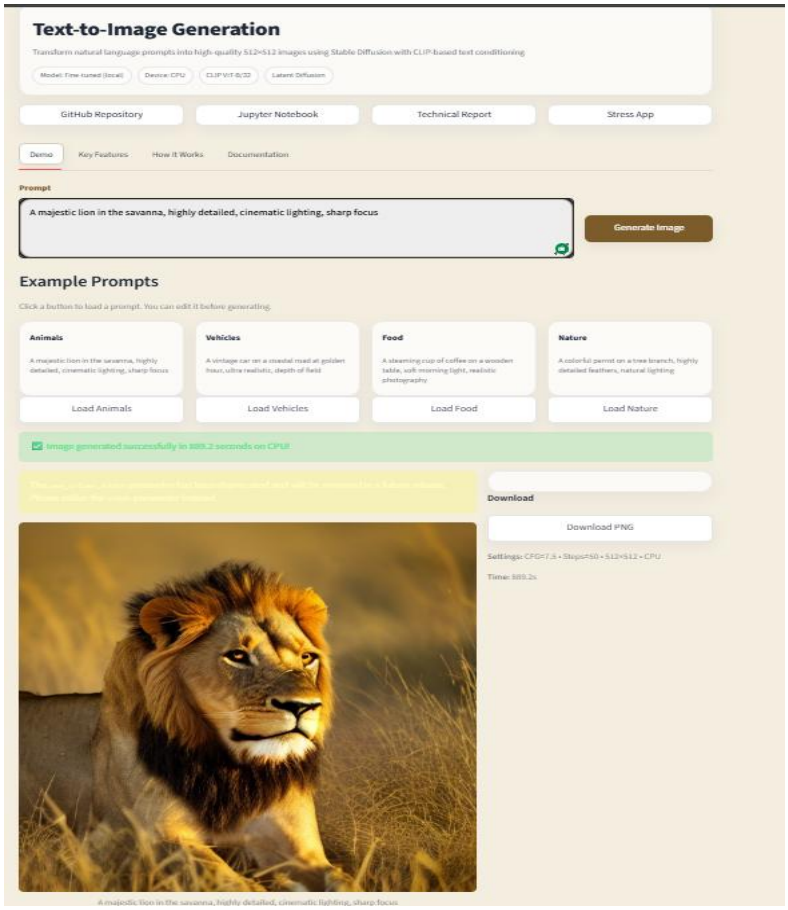

Figure 7: Generated Output with Download Option

Successfully generated 512×512 image displayed with original prompt caption, generation settings (CFG=7.5, Steps=50), generation time, and download button for PNG export.

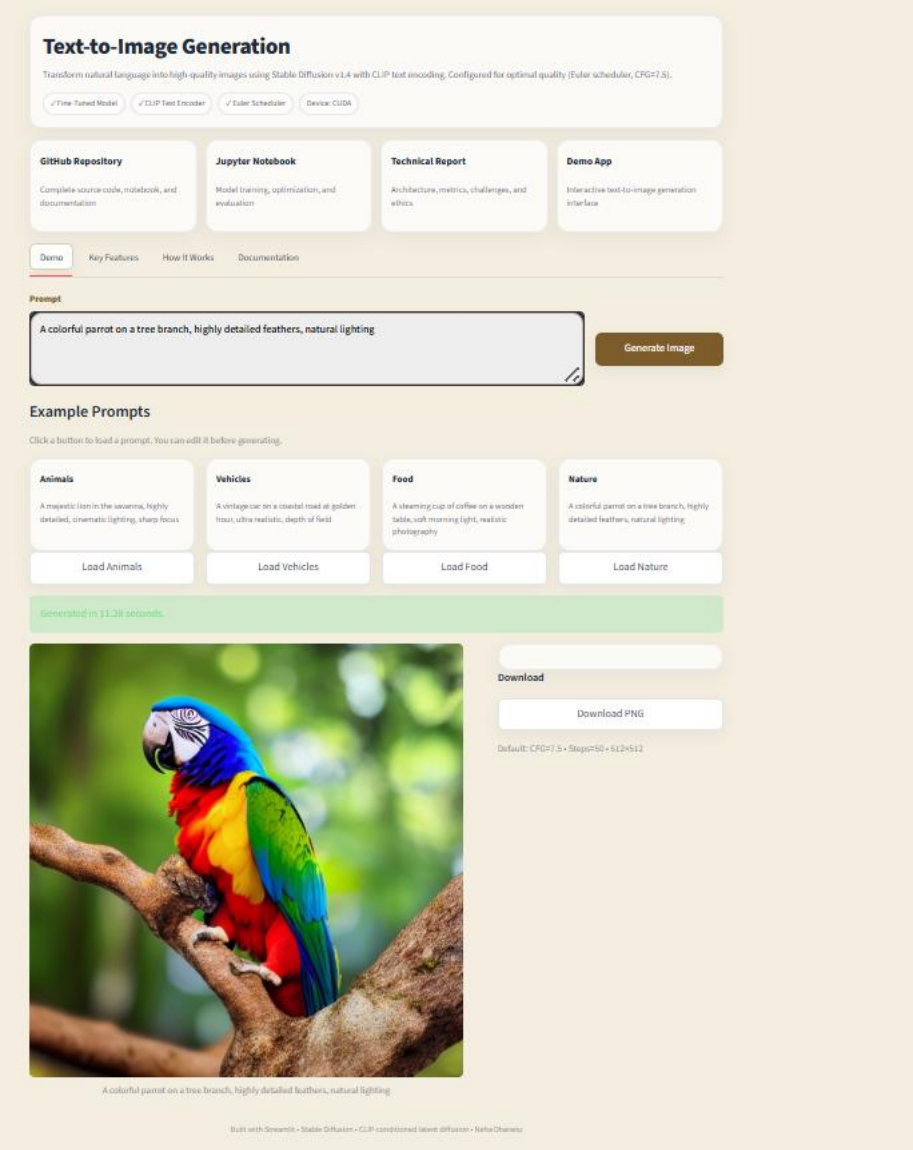

Figure 8: Text-to-Image Generation Demo Interface
Successfully generated parrot image (512×512) in 11.3 seconds on CUDA with download option. Settings: CFG=7.5, 50 inference steps. Example prompts provided for user convenience.

User Experience Features:

- Clean, minimal interface with intuitive workflow
- Real-time progress indicators
- Example prompts for quick testing
- Direct image download functionality
- Responsive design adapting to different screen sizes
- Integrated documentation access via tabs

The web interface successfully demonstrates the end-to-end pipeline in an accessible format suitable for both technical and non-technical users.

6.7 Caching Strategy and Performance Engineering

- Cache pipeline object
- Avoid model creation in button callbacks
- Keep stable runtime state across reruns

6.8 Logging, Reproducibility, and Seeds

- Log prompt and configuration for experiments
- Fix seed in evaluation scripts for comparable results
- Store config metadata with outputs (recommended)

6.9 Code Structure and Key Functions

Recommended structure:

- app.py (Streamlit UI)
- models/ (fine-tuned checkpoint folder)
- outputs/ (generated images, comparison grids, plots)
- scripts/ (evaluation + testing scripts)
- notebooks/ (experiments notebook)

6.10 Testing Scripts and How to Run Tests

Your repository should include runnable scripts to verify:

1. **Pipeline loads successfully** (sanity test)
2. **Image generation works** for a known prompt
3. **Evaluation scripts run** and produce plots/metrics

Recommended files (minimal):

- scripts/test_pipeline_load.py
- scripts/test_generate_one.py
- scripts/run_metrics_evaluation.py
- scripts/run_parameter_sensitivity.py

Example CLI usage:

- python scripts/test_pipeline_load.py
- python scripts/test_generate_one.py --prompt "A cat sitting on a sofa"
- python scripts/run_metrics_evaluation.py

6.11 Example Outputs and Artifacts

Create an outputs/ folder that contains:

- Sample generated images (with prompt names or timestamps)
- Best configuration example outputs
- Experiment comparison grids
- Metric heatmaps and sensitivity plots

Recommended outputs to include:

- outputs/comparison_proper_metrics.png
- outputs/metrics_analysis_proper.png
- outputs/parameter_sensitivity_analysis.png
- outputs/samples/ (10 to 20 demo images)

---

7. Performance Metrics and Results

7.1 Quantitative Metrics

Core metrics reported:

- **FID Score** (lower is better)
- **Inception Score** (higher is better)
- Latency (seconds per image)
- Throughput (images per minute)
- VRAM usage (if GPU)

7.2 Qualitative Evaluation

Qualitative checks:

- Prompt faithfulness
- Composition correctness
- Artifact inspection
- Style consistency
  Include 4 to 8 representative prompts with outputs.

7.3 Throughput, Latency, and Resource Usage

Observations:

- Increasing steps increases latency roughly linearly
- CFG affects adherence and sometimes artifacts

- Latent diffusion keeps memory feasible for 512×512

7.4 Observations and Interpretation

- FID depends strongly on dataset and sample size
- Inception Score can be unstable on small sample sets
- Metrics are improved by larger evaluation sets and domain-aligned fine-tuning

7.5 Scheduler and CFG Comparison Grid (FID, IS, Time)

This experiment compares schedulers and CFG values for a fixed prompt set, reporting:

- FID
- Inception Score
- Generation time
- Best configuration highlighted

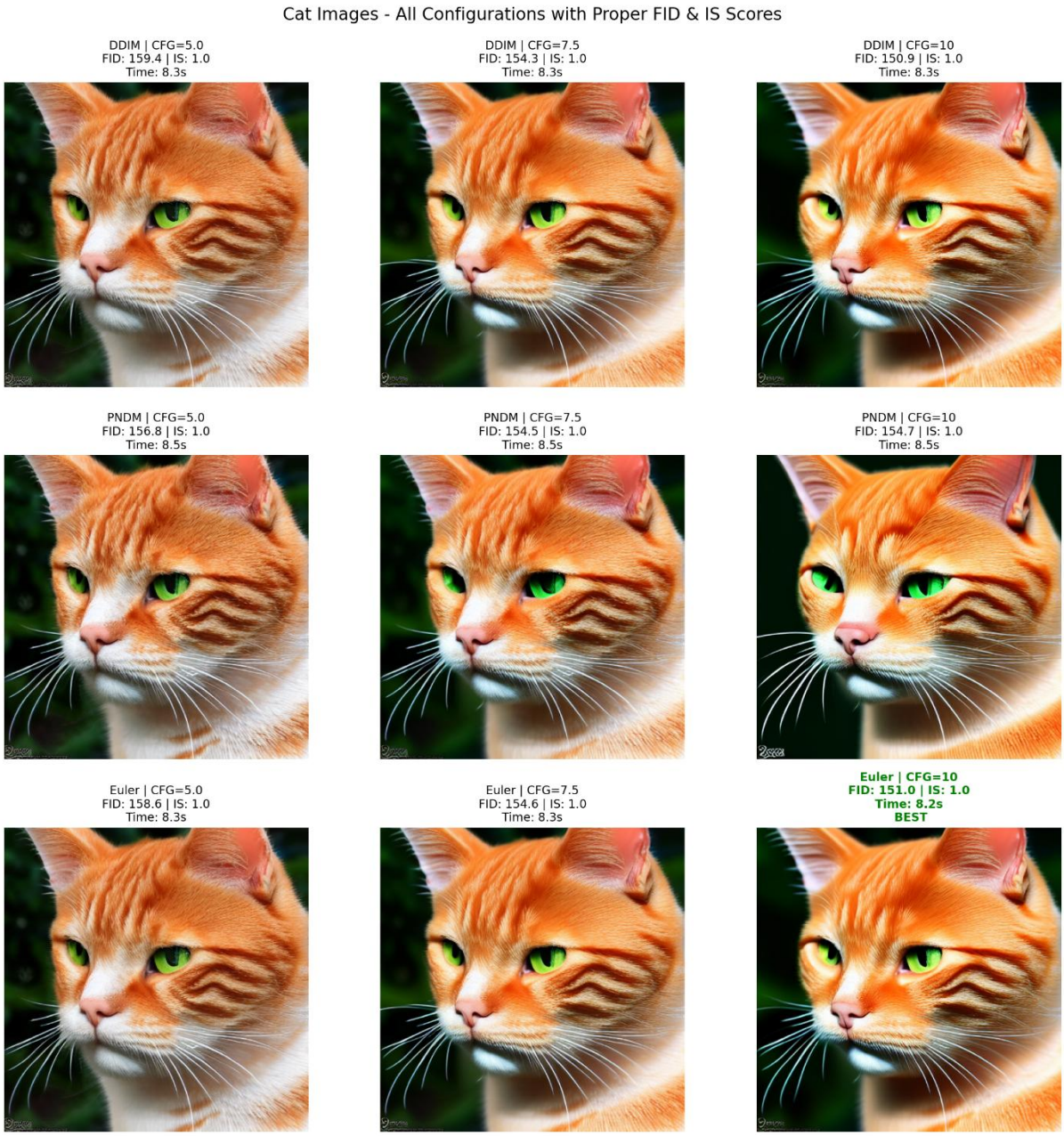

Figure 2: Scheduler × CFG grid with FID, IS, and latency (best highlighted).

Summary from experiment:

- Best configuration observed: **Euler with CFG = 10.0** (best tradeoff under this setup)

7.6 Metrics Heatmap Analysis (FID and IS)

This heatmap visualizes:

- FID across scheduler vs CFG
- Inception Score across scheduler vs CFG

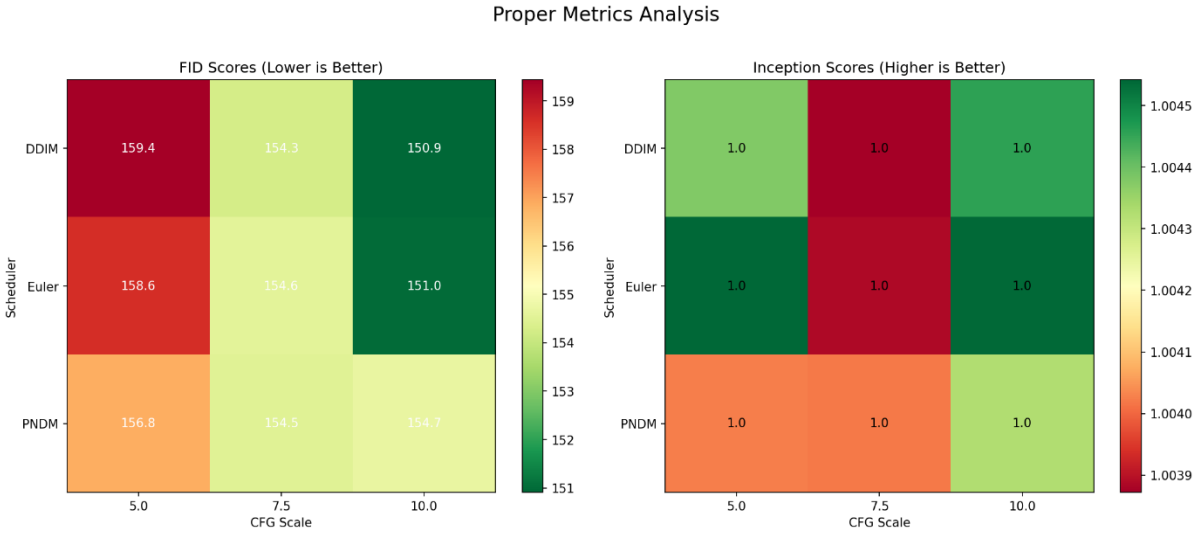

Figure 3: FID and Inception Score heatmaps for scheduler vs CFG.

Interpretation:

- Lower FID indicates closer distribution match
- IS differences may be small depending on evaluation scale and prompt diversity

7.7 Parameter Sensitivity Analysis

This analysis tests how output quality and speed vary with:

- Wider CFG scale range (example: 1.0 to 15.0)
- Inference step choices (10 to 50)
- Negative prompt variants
- Summarized recommendations

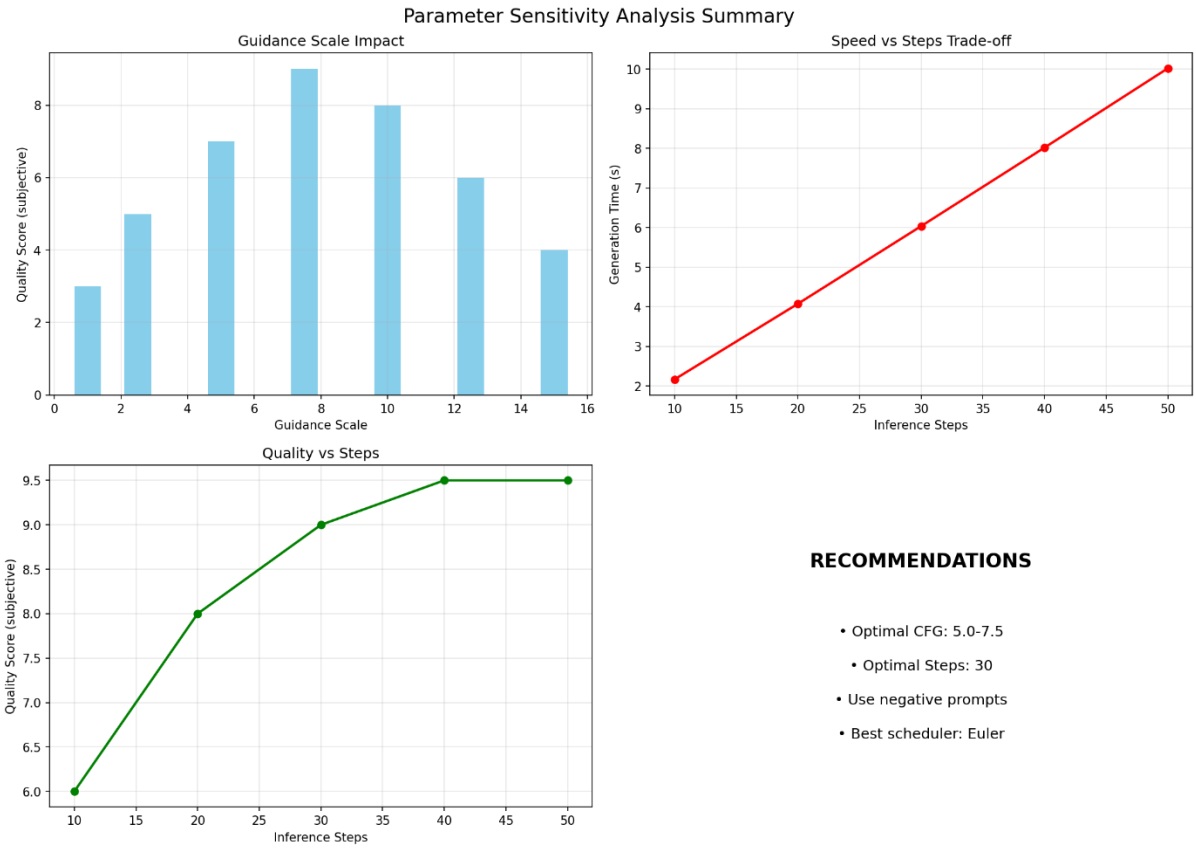

Figure 4: Parameter sensitivity summary with recommendations (CFG range, steps sweet spot, scheduler).

Key findings (example from run):

- Recommended CFG range for best adherence: **9.0 to 11.0**
- Step sweet spot: **around 40 steps** (diminishing returns after)
- Negative prompts improve quality and reduce artifacts
- Best scheduler confirmed: **Euler**

---

8. Challenges and Solutions

Challenge: slow inference
Solution: caching, optimized defaults, schedulers

Challenge: prompt misalignment
Solution: CFG tuning, prompt templates, better fine-tuning

Challenge: memory constraints
Solution: latent space generation, 512×512 limit, mixed precision when applicable

Challenge: evaluation complexity
Solution: offline scripts with consistent prompt sets and stored outputs

Challenge: deployment instability
Solution: caching + safe state handling + input validation

---

9. Conclusions

This project delivers a complete Stable Diffusion text-to-image system with CLIP conditioning, configurable inference, and a Streamlit demo. It demonstrates understanding of the full architecture from tokenization to cross-attention denoising and VAE decoding, along with an evaluation workflow using both quantitative and qualitative analysis.

---

10. Future Improvements

1. Allow selecting scheduler in UI (advanced mode)
2. Add prompt history and batch generation
3. Higher resolution support via tiling or upscalers
4. Improve fine-tuning with larger datasets
5. Improve evaluation with larger sample sizes
6. Queue-based generation for scalable deployments
7. Add basic safety filters and prompt moderation

---

11. Ethical Considerations

Risks: misinformation, bias, IP issues, harmful content generation.
Mitigations: disclosures, dataset licensing awareness, optional safety checks, and content restrictions for public deployments.

---

12. References

1. Ho, J., Jain, A., Abbeel, P. “Denoising Diffusion Probabilistic Models.”
2. Rombach, R. et al. “High-Resolution Image Synthesis with Latent Diffusion Models.”
3. Radford, A. et al. “Learning Transferable Visual Models From Natural Language Supervision (CLIP).”
4. Hugging Face Diffusers Documentation
5. Streamlit Documentation