```python
# Step 1: Importing necessasery libraries

from datasets import load_dataset
from PIL import Image
import pandas as pd
import matplotlib.pyplot as plt
import re, random

import os, glob


#  Step 2: Load subset of the dataset
ds = load_dataset("ashraq/fashion-product-images-small",
split="train[:4000]")
```

```
/usr/local/lib/python3.12/dist-packages/huggingface_hub/utils/
_auth.py:94: UserWarning:
The secret `HF_TOKEN` does not exist in your Colab secrets.
To authenticate with the Hugging Face Hub, create a token in your
settings tab (https://huggingface.co/settings/tokens), set it as
secret in your Google Colab and restart your session.
You will be able to reuse this secret in all of your notebooks.
Please note that authentication is recommended but still optional to
access public models or datasets.
  warnings.warn(
```


```
{"model_id":"ec7ab17438e34bcdb77cd5b29078d60c","version_major":2,"vers
ion_minor":0}

{"model_id":"d7247036a96e4369928e72f5ff2c2fcd","version_major":2,"vers
ion_minor":0}

{"model_id":"c5b01edc497443f7b07908e339e4a044","version_major":2,"vers
ion_minor":0}

{"model_id":"451be63a875245a48038cfa66eb50e0c","version_major":2,"vers
ion_minor":0}
```


```python
# Basic summary
print("Dataset Summary")
print("-----------------")
ds
print("\nColumns:", ds.column_names)
print("Total samples:", len(ds))
```

```
Dataset Summary
-----------------

Columns: ['id', 'gender', 'masterCategory', 'subCategory',
'articleType', 'baseColour', 'season', 'year', 'usage',
```

```
'productDisplayName', 'image']
Total samples: 4000
```

```
ds.info
```

```
DatasetInfo(description='', citation='', homepage='', license='',
features={'id': Value('int64'), 'gender': Value('string'),
'masterCategory': Value('string'), 'subCategory': Value('string'),
'articleType': Value('string'), 'baseColour': Value('string'),
'season': Value('string'), 'year': Value('float64'), 'usage':
Value('string'), 'productDisplayName': Value('string'), 'image':
Image(mode=None, decode=True)}, post_processed=None,
supervised_keys=None, builder_name='parquet', dataset_name='fashion-
product-images-small', config_name='default', version=0.0.0,
splits={'train': SplitInfo(name='train', num_bytes=568896653,
num_examples=44072, shard_lengths=[42036, 2036],
dataset_name='fashion-product-images-small')},
download_checksums={'hf://datasets/ashraq/fashion-product-images-
small@3859c76db2f6f3d3b9a3863345e3ccdbff75879d/data/train-00000-of-
00002-6cff4c59f91661c3.parquet': {'num_bytes': 136091680, 'checksum':
None}, 'hf://datasets/ashraq/fashion-product-images-
small@3859c76db2f6f3d3b9a3863345e3ccdbff75879d/data/train-00001-of-
00002-bb459e5ac5f01e71.parquet': {'num_bytes': 135404761, 'checksum':
None}}, download_size=271496441, post_processing_size=None,
dataset_size=568896653, size_in_bytes=840393094)
```

```
# Check column data types
print("\nColumn types:")
ds.features
```

```
Column types:
```

```
{'id': Value('int64'),
 'gender': Value('string'),
 'masterCategory': Value('string'),
 'subCategory': Value('string'),
 'articleType': Value('string'),
 'baseColour': Value('string'),
 'season': Value('string'),
 'year': Value('float64'),
 'usage': Value('string'),
 'productDisplayName': Value('string'),
 'image': Image(mode=None, decode=True)}
```

```
# Peek into first few rows
print("\nSample rows:")
ds[:3]
```

```
Sample rows:
```

```
{'id': [15970, 39386, 59263],
 'gender': ['Men', 'Men', 'Women'],
 'masterCategory': ['Apparel', 'Apparel', 'Accessories'],
 'subCategory': ['Topwear', 'Bottomwear', 'Watches'],
 'articleType': ['Shirts', 'Jeans', 'Watches'],
 'baseColour': ['Navy Blue', 'Blue', 'Silver'],
 'season': ['Fall', 'Summer', 'Winter'],
 'year': [2011.0, 2012.0, 2016.0],
 'usage': ['Casual', 'Casual', 'Casual'],
 'productDisplayName': ['Turtle Check Men Navy Blue Shirt',
  'Peter England Men Party Blue Jeans',
  'Titan Women Silver Watch'],
 'image': [<PIL.JpegImagePlugin.JpegImageFile image mode=RGB
size=60x80>,
  <PIL.JpegImagePlugin.JpegImageFile image mode=RGB size=60x80>,
  <PIL.Image.Image image mode=L size=60x80>]}

#  Step 3: Text cleaning helper
def clean_caption(t):
    if t is None:
        return ""
    t = re.sub(r"\s+", " ", str(t).lower().strip())
    return t[:120]  # truncate long captions

#  Step 4: Convert dataset into a DataFrame
records = []
for rec in ds:
    img = rec["image"]
    text = clean_caption(rec["productDisplayName"])
    records.append({
        "text": text,
        "category": rec["masterCategory"],
        "subcategory": rec["subCategory"],
        "gender": rec["gender"],
        "image": img
    })

df = pd.DataFrame(records)
print(f"Loaded {len(df)} valid (image, text) pairs.")
print(df.head())

Loaded 4000 valid (image, text) pairs.
                                              text     category
subcategory  \
0               turtle check men navy blue shirt      Apparel
Topwear
1             peter england men party blue jeans      Apparel
Bottomwear
2                        titan women silver watch  Accessories
Watches
```

```
3  manchester united men solid black track pants      Apparel
Bottomwear
4                          puma men grey t-shirt      Apparel
Topwear

  gender                                                image
0    Men  <PIL.JpegImagePlugin.JpegImageFile image mode=...
1    Men  <PIL.JpegImagePlugin.JpegImageFile image mode=...
2  Women  <PIL.Image.Image image mode=L size=60x80 at 0x...
3    Men  <PIL.JpegImagePlugin.JpegImageFile image mode=...
4    Men  <PIL.Image.Image image mode=RGB size=60x80 at ...
```

```python
#  Step 5: Visualize random samples
import matplotlib.pyplot as plt
from textwrap import fill

def show_samples(df, n=5):
    samples = df.sample(n).reset_index(drop=True)
    plt.figure(figsize=(4*n, 5))

    for i, (_, row) in enumerate(samples.iterrows()):
        plt.subplot(1, n, i + 1)
        plt.imshow(row["image"])
        plt.axis("off")

        # wrap and shorten title text
        title = fill(str(row["text"])[:80], width=20)
        plt.title(title, fontsize=9, pad=8)

    plt.tight_layout()
    plt.show()

show_samples(df)
```


```python
#  Step 6: Basic stats
print("\n Category distribution:")
print(df["category"].value_counts().head(10))

print("\n Gender distribution:")
```

```
print(df["gender"].value_counts())

# Save the cleaned version
df.to_pickle("fashion_cleaned.pkl")


 Category distribution:
category
Apparel          1844
Accessories      1075
Footwear          862
Personal Care     207
Free Items         12
Name: count, dtype: int64

 Gender distribution:
gender
Men       1970
Women     1721
Unisex     176
Girls       68
Boys        65
Name: count, dtype: int64

# Install Hugging Face diffusers + CLIP
!pip install -q diffusers transformers torch torchvision pillow

import torch
from transformers import CLIPProcessor, CLIPModel
from diffusers import StableDiffusionPipeline
import matplotlib.pyplot as plt

# Load CLIP model (for embeddings)
clip_model = CLIPModel.from_pretrained("openai/clip-vit-base-patch32")
clip_processor = CLIPProcessor.from_pretrained("openai/clip-vit-base-
patch32")

# Load a small Stable Diffusion model for quick demo
sd_pipe = StableDiffusionPipeline.from_pretrained(
    "runwayml/stable-diffusion-v1-5",
    torch_dtype=torch.float16 if torch.cuda.is_available() else
torch.float32
)
sd_pipe = sd_pipe.to("cuda" if torch.cuda.is_available() else "cpu")

print(" Environment ready (CLIP + Diffusers initialized)")

{"model_id":"61a8587fdbc64512a7ac890720f1490b","version_major":2,"vers
ion_minor":0}
```

{"model_id":"1f9a8ab580d747b6a362d3030a8e1e7a","version_major":2,"version_minor":0}

Using a slow image processor as `use_fast` is unset and a slow processor was saved with this model. `use_fast=True` will be the default behavior in v4.52, even if the model was saved with a slow processor. This will result in minor differences in outputs. You'll still be able to use a slow processor with `use_fast=False`.

{"model_id":"884bf523764f46469cb57d956c7feb14","version_major":2,"version_minor":0}

{"model_id":"dcd9cee76b75497e83415e85e211736a","version_major":2,"version_minor":0}

{"model_id":"08310e43bb1a4265b0fabe3aa583a6fb","version_major":2,"version_minor":0}

{"model_id":"c942635319194bbea98b08dd39654ff9","version_major":2,"version_minor":0}

{"model_id":"e37e5a0d8ce34f8e922600c0c417644e","version_major":2,"version_minor":0}

{"model_id":"ada4ca669ea540af9603ef790633d282","version_major":2,"version_minor":0}

{"model_id":"09ca1936d1384c1f919871d56db288c9","version_major":2,"version_minor":0}

{"model_id":"bf80618296c34f2b829b1dad15f9ba30","version_major":2,"version_minor":0}

{"model_id":"667091c56fc945039650a9331ab25158","version_major":2,"version_minor":0}

{"model_id":"05863ec364fa4b3b9d56df742868bd9a","version_major":2,"version_minor":0}

{"model_id":"11734b43d33742d18977ffc8a15e6ed3","version_major":2,"version_minor":0}

{"model_id":"0f3bd3df520f4d56a875775f23244de6","version_major":2,"version_minor":0}

{"model_id":"a38e603f02c644448e2052c63425909c","version_major":2,"version_minor":0}

{"model_id":"f8aa622be2914a01afab564b211321e4","version_major":2,"version_minor":0}

{"model_id":"e95deb9f2c1d462abb8ad6fa53d0ea07","version_major":2,"version_minor":0}

{"model_id":"ef689fc1bc6a4505bf36fc6932acb7ba","version_major":2,"version_minor":0}

{"model_id":"ed6e736a0b0146d09d8d5e20c88fd8b2","version_major":2,"version_minor":0}

{"model_id":"9fb5171d11b5487cb84a94cbee6d3dd3","version_major":2,"version_minor":0}

{"model_id":"dd4b421b1966450494a085196206344c","version_major":2,"version_minor":0}

{"model_id":"e33f8922e4f74102988caac931f699d9","version_major":2,"version_minor":0}

{"model_id":"01a33c3a1c804dc7a213154217a708f2","version_major":2,"version_minor":0}

{"model_id":"15b08090e2a74b78bdd255aa93d38275","version_major":2,"version_minor":0}

{"model_id":"17d15fa8a5654a69925596fd0030f1df","version_major":2,"version_minor":0}

{"model_id":"dd5593ee4e194166937fa5adcc18dc5d","version_major":2,"version_minor":0}

```
`torch_dtype` is deprecated! Use `dtype` instead!
```

```
 Environment ready (CLIP + Diffusers initialized)
```

```
# Taking 5 random product descriptions

import pandas as pd
df = pd.read_pickle("fashion_cleaned.pkl")
print(" Reloaded DataFrame:", df.shape)

sample_texts = df["text"].sample(5).tolist()

inputs = clip_processor(
    text=sample_texts,
    return_tensors="pt",
    padding=True
)
with torch.no_grad():
    text_embeds = clip_model.get_text_features(**inputs)

print(" Generated CLIP text embeddings with shape:",
text_embeds.shape)
```

```
 Reloaded DataFrame: (4000, 5)
 Generated CLIP text embeddings with shape: torch.Size([5, 512])
```

```
# Generate 5 prompt→image outputs
for i, prompt in enumerate(sample_texts):
    image = sd_pipe(prompt).images[0]
    plt.figure(figsize=(3,3))
    plt.imshow(image)
    plt.axis("off")
    plt.title(prompt[:50]+"...")
    plt.show()
```

{"model_id":"3276f6f75d904529b80899fb9dbc36c3","version_major":2,"version_minor":0}

ucb men's johny collar with two tone grey t-shirt...

{"model_id":"c9f2a36dcfc34dd0b007742e41441b5b","version_major":2,"version_minor":0}

puma men red ferrari t-shirt...

{"model_id":"e07bb844d79f41d9991b4bfe002ed287","version_major":2,"version_minor":0}

nike men air max conquer acg olive sports boot...

{"model_id":"0b72d5db6f9d4c3780bec1f84c51b735","version_major":2,"version_minor":0}

prafful multi coloured sari...

{"model_id":"c7888703639e4303adcc098fbc585b97","version_major":2,"version_minor":0}

gini and jony kids solid black capris...

```
import torch
from diffusers import DDIMScheduler, EulerDiscreteScheduler,
PNDMScheduler
import os
import itertools

# --- 1. Setup ---

# Ensure an output directory exists
output_dir = "m2_early_samples"
os.makedirs(output_dir, exist_ok=True)
device = "cuda" if torch.cuda.is_available() else "cpu"

# --- 2. Define Prompts and Parameters ---

# Using 3 of the sample prompts
prompts = {
    "watch": "titan women silver watch",
    "shirt": "adidas mens fire grey t-shirt",
    "sweatshirt": "proline men green print ed sweatshirt"
}

# Define the parameters to test
param_grid = {
    "guidance_scale": [7.5, 9.0, 12.0],
    "num_inference_steps": [30, 50],
    "scheduler": [
        ("PNDM", PNDMScheduler.from_config(sd_pipe.scheduler.config)),
        ("DDIM", DDIMScheduler.from_config(sd_pipe.scheduler.config)),
        ("Euler",
EulerDiscreteScheduler.from_config(sd_pipe.scheduler.config))
    ]
}
```

```
# --- 3. Run Experiment Loop ---

print("Starting Milestone 2 inference tuning...")
print(f"Saving images to: {output_dir}/")

log_entries = []

# Iterate over each prompt
for prompt_key, prompt_text in prompts.items():
    print(f"\n--- Processing prompt: '{prompt_text}' ---")

    # Iterate over each combination of parameters
    for guidance, steps, (scheduler_name, scheduler_obj) in
itertools.product(
        param_grid["guidance_scale"],
        param_grid["num_inference_steps"],
        param_grid["scheduler"]
    ):

        # Set the pipeline's scheduler
        sd_pipe.scheduler = scheduler_obj

        # Generate a unique filename
        filename =
f"{prompt_key}_g{guidance}_s{steps}_{scheduler_name}.png"
        output_path = os.path.join(output_dir, filename)

        log_msg = f"Generating: {filename}"
        print(log_msg)

        # Generate the image
        with torch.no_grad():
            image = sd_pipe(
                prompt_text,
                guidance_scale=guidance,
                num_inference_steps=steps
            ).images[0]

        # Save the image
        image.save(output_path)

        # Add to log
        log_entries.append(log_msg)

print("\n--- Experiment complete ---")
print(f"Generated {len(log_entries)} sample images in
'{output_dir}'.")
```

```
Starting Milestone 2 inference tuning...
Saving images to: m2_early_samples/
```

```
--- Processing prompt: 'titan women silver watch' ---
Generating: watch_g7.5_s30_PNDM.png

{"model_id":"077b87275abc40c9aa4a4d970a20cda9","version_major":2,"vers
ion_minor":0}

Generating: watch_g7.5_s30_DDIM.png

{"model_id":"1e781daa633146ceb9638769d81eedfa","version_major":2,"vers
ion_minor":0}

Generating: watch_g7.5_s30_Euler.png

{"model_id":"508ca4a680f741cab45201100fa091e3","version_major":2,"vers
ion_minor":0}

Generating: watch_g7.5_s50_PNDM.png

{"model_id":"9a3347315e1d4bb18506446cb04b0b04","version_major":2,"vers
ion_minor":0}

Generating: watch_g7.5_s50_DDIM.png

{"model_id":"1bbe8a54a0fb4a5190fb58fb74f7f9b0","version_major":2,"vers
ion_minor":0}

Generating: watch_g7.5_s50_Euler.png

{"model_id":"2d45d422b2014ae78168f20a95b4ac98","version_major":2,"vers
ion_minor":0}

Generating: watch_g9.0_s30_PNDM.png

{"model_id":"79d674afe6654e70bf06f9fff4597b85","version_major":2,"vers
ion_minor":0}

Generating: watch_g9.0_s30_DDIM.png

{"model_id":"425e18aa26974dfb85e94524c3a58dc0","version_major":2,"vers
ion_minor":0}

Generating: watch_g9.0_s30_Euler.png

{"model_id":"d1ed9523badf431ab5cca4ce6a5935cb","version_major":2,"vers
ion_minor":0}

Generating: watch_g9.0_s50_PNDM.png

{"model_id":"9b4b37b856bc46daa9e9d640135987a8","version_major":2,"vers
ion_minor":0}

Generating: watch_g9.0_s50_DDIM.png
```

{"model_id":"2db5259ae3024fc1ba1e97e043806dac","version_major":2,"version_minor":0}

```
Generating: watch_g9.0_s50_Euler.png
```

{"model_id":"d8c70b1e6601464eab42811c478a160c","version_major":2,"version_minor":0}

```
Generating: watch_g12.0_s30_PNDM.png
```

{"model_id":"76f909c1b55b4d48b9265387969c8019","version_major":2,"version_minor":0}

```
Generating: watch_g12.0_s30_DDIM.png
```

{"model_id":"aae8a419c5694f64a71243df084f0306","version_major":2,"version_minor":0}

```
Generating: watch_g12.0_s30_Euler.png
```

{"model_id":"ec8ff9cd44a74054a85411c500d1131c","version_major":2,"version_minor":0}

```
Generating: watch_g12.0_s50_PNDM.png
```

{"model_id":"695cfdb3e9154665b253cb502cfbfd59","version_major":2,"version_minor":0}

```
Generating: watch_g12.0_s50_DDIM.png
```

{"model_id":"6f38aab189374de4b78439b92da2df01","version_major":2,"version_minor":0}

```
Generating: watch_g12.0_s50_Euler.png
```

{"model_id":"b02d8369d58045c39b15442ac0af06fd","version_major":2,"version_minor":0}

```
--- Processing prompt: 'adidas mens fire grey t-shirt' ---
Generating: shirt_g7.5_s30_PNDM.png
```

{"model_id":"34ec119802c146eaa73df37e86fbfe6d","version_major":2,"version_minor":0}

```
Generating: shirt_g7.5_s30_DDIM.png
```

{"model_id":"9b0b7525f19c4b51a9a0124d16f064d6","version_major":2,"version_minor":0}

```
Generating: shirt_g7.5_s30_Euler.png
```

{"model_id":"e01549bf90204132b95b58b858376abc","version_major":2,"version_minor":0}

```
Generating: shirt_g7.5_s50_PNDM.png

{"model_id":"1904003b911b42ad9584ca5961e3dab7","version_major":2,"vers
ion_minor":0}

Generating: shirt_g7.5_s50_DDIM.png

{"model_id":"0f215f70148143f68b947627d714c23c","version_major":2,"vers
ion_minor":0}

Generating: shirt_g7.5_s50_Euler.png

{"model_id":"5fb3e8baad6d4024a1f2c38167b1db13","version_major":2,"vers
ion_minor":0}

Generating: shirt_g9.0_s30_PNDM.png

{"model_id":"2ef1ae5b28ba47b9a54adf314ad1ea82","version_major":2,"vers
ion_minor":0}

Generating: shirt_g9.0_s30_DDIM.png

{"model_id":"777ee79fa08d4a88ab91674298703f2d","version_major":2,"vers
ion_minor":0}

Generating: shirt_g9.0_s30_Euler.png

{"model_id":"4f7e303083fd490f9a5e7e355b7dd2c5","version_major":2,"vers
ion_minor":0}

Generating: shirt_g9.0_s50_PNDM.png

{"model_id":"f8becb0906d143c48ecd7f3ab4f08a36","version_major":2,"vers
ion_minor":0}

Generating: shirt_g9.0_s50_DDIM.png

{"model_id":"c68649e5cfda4147bb3a88f00a6fc06f","version_major":2,"vers
ion_minor":0}

Generating: shirt_g9.0_s50_Euler.png

{"model_id":"a0198a2e77d645e2934cd17d6435f473","version_major":2,"vers
ion_minor":0}

Generating: shirt_g12.0_s30_PNDM.png

{"model_id":"2f05d897a9634010b2a848fdd4b6c332","version_major":2,"vers
ion_minor":0}

Generating: shirt_g12.0_s30_DDIM.png

{"model_id":"4582fa52555d457cad4a776771c3f20e","version_major":2,"vers
ion_minor":0}
```

```
Generating: shirt_g12.0_s30_Euler.png

{"model_id":"bafce18f8da54abf81785afb9bc56890","version_major":2,"vers
ion_minor":0}

Generating: shirt_g12.0_s50_PNDM.png

{"model_id":"b8684cd1ed5b446d9e295d070c797fb6","version_major":2,"vers
ion_minor":0}

Generating: shirt_g12.0_s50_DDIM.png

{"model_id":"378bc3fa91cd44afb564b4da36e699d0","version_major":2,"vers
ion_minor":0}

Generating: shirt_g12.0_s50_Euler.png

{"model_id":"b4726152a3fd4c62be5ab789d34788cf","version_major":2,"vers
ion_minor":0}


--- Processing prompt: 'proline men green print ed sweatshirt' ---
Generating: sweatshirt_g7.5_s30_PNDM.png

{"model_id":"7747f19294ca490a9de76b23383e95d7","version_major":2,"vers
ion_minor":0}

Generating: sweatshirt_g7.5_s30_DDIM.png

{"model_id":"6d4485ce24c943738e2f18729bbf2e7e","version_major":2,"vers
ion_minor":0}

Generating: sweatshirt_g7.5_s30_Euler.png

{"model_id":"604be2b44e1e4c0888f0eec0e82c737f","version_major":2,"vers
ion_minor":0}

Generating: sweatshirt_g7.5_s50_PNDM.png

{"model_id":"793cda5d2aa7412ebc2948bc5b1e5983","version_major":2,"vers
ion_minor":0}

Generating: sweatshirt_g7.5_s50_DDIM.png

{"model_id":"0cac7d76c8df40aebc179397dfbfd7d0","version_major":2,"vers
ion_minor":0}

Generating: sweatshirt_g7.5_s50_Euler.png

{"model_id":"4a2c29c8aff049ca8fd70143bfdbb591","version_major":2,"vers
ion_minor":0}

Generating: sweatshirt_g9.0_s30_PNDM.png
```

{"model_id":"90e5e3cd268849c1b0d0d9f3b47711ad","version_major":2,"version_minor":0}

```
Generating: sweatshirt_g9.0_s30_DDIM.png
```

{"model_id":"7f6ddcfe2e964c09ae80e15c13c6a2b2","version_major":2,"version_minor":0}

```
Generating: sweatshirt_g9.0_s30_Euler.png
```

{"model_id":"d26e20853da245b0bcd23b784872c63b","version_major":2,"version_minor":0}

```
Generating: sweatshirt_g9.0_s50_PNDM.png
```

{"model_id":"d31f2cc396614d32b89e1d3ec4213173","version_major":2,"version_minor":0}

```
Generating: sweatshirt_g9.0_s50_DDIM.png
```

{"model_id":"4bb307e8328f445099d6c178feae34ce","version_major":2,"version_minor":0}

```
Generating: sweatshirt_g9.0_s50_Euler.png
```

{"model_id":"de722136a7be400e8716c7d48ff675a0","version_major":2,"version_minor":0}

```
Generating: sweatshirt_g12.0_s30_PNDM.png
```

{"model_id":"12bda29f4ead408dac33af5c1c7f4cb5","version_major":2,"version_minor":0}

```
Generating: sweatshirt_g12.0_s30_DDIM.png
```

{"model_id":"a2facd77ba514ec0a78bdf7995a6dba9","version_major":2,"version_minor":0}

```
Generating: sweatshirt_g12.0_s30_Euler.png
```

{"model_id":"0bfd420a22b844a18e745015dd819ceb","version_major":2,"version_minor":0}

```
Generating: sweatshirt_g12.0_s50_PNDM.png
```

{"model_id":"ec8dc1f2861f4c5db423af68428fa878","version_major":2,"version_minor":0}

```
Generating: sweatshirt_g12.0_s50_DDIM.png
```

{"model_id":"c0403495290b434eae633adf5eccc15e","version_major":2,"version_minor":0}

```
Generating: sweatshirt_g12.0_s50_Euler.png
```

{"model_id":"bbf0e801e0654a2f8988ecb4578f0b0b","version_major":2,"version_minor":0}

```
--- Experiment complete ---
Generated 54 sample images in 'm2_early_samples'.
```

```
import matplotlib.pyplot as plt
from PIL import Image
import glob

# Get list of generated images
image_paths = glob.glob("m2_early_samples/*.png")[:10] # Show first 5

plt.figure(figsize=(20, 5))
for i, img_path in enumerate(image_paths):
    img = Image.open(img_path)
    plt.subplot(1, 10, i+1)
    plt.imshow(img)
    plt.axis("off")
    plt.title(os.path.basename(img_path), fontsize=8)
plt.show()
```


```
!pip install pytorch-fid scipy
!pip install git+https://github.com/openai/CLIP.git
```

```
Collecting pytorch-fid
  Downloading pytorch_fid-0.3.0-py3-none-any.whl.metadata (5.3 kB)
Requirement already satisfied: scipy in
/usr/local/lib/python3.12/dist-packages (1.16.3)
Requirement already satisfied: numpy in
/usr/local/lib/python3.12/dist-packages (from pytorch-fid) (2.0.2)
Requirement already satisfied: pillow in
/usr/local/lib/python3.12/dist-packages (from pytorch-fid) (11.3.0)
Requirement already satisfied: torch>=1.0.1 in
/usr/local/lib/python3.12/dist-packages (from pytorch-fid)
(2.9.0+cu126)
Requirement already satisfied: torchvision>=0.2.2 in
/usr/local/lib/python3.12/dist-packages (from pytorch-fid)
(0.24.0+cu126)
Requirement already satisfied: filelock in
/usr/local/lib/python3.12/dist-packages (from torch>=1.0.1->pytorch-
fid) (3.20.0)
Requirement already satisfied: typing-extensions>=4.10.0 in
/usr/local/lib/python3.12/dist-packages (from torch>=1.0.1->pytorch-
```

```
fid) (4.15.0)
Requirement already satisfied: setuptools in
/usr/local/lib/python3.12/dist-packages (from torch>=1.0.1->pytorch-
fid) (75.2.0)
Requirement already satisfied: sympy>=1.13.3 in
/usr/local/lib/python3.12/dist-packages (from torch>=1.0.1->pytorch-
fid) (1.14.0)
Requirement already satisfied: networkx>=2.5.1 in
/usr/local/lib/python3.12/dist-packages (from torch>=1.0.1->pytorch-
fid) (3.6)
Requirement already satisfied: jinja2 in
/usr/local/lib/python3.12/dist-packages (from torch>=1.0.1->pytorch-
fid) (3.1.6)
Requirement already satisfied: fsspec>=0.8.5 in
/usr/local/lib/python3.12/dist-packages (from torch>=1.0.1->pytorch-
fid) (2025.3.0)
Requirement already satisfied: nvidia-cuda-nvrtc-cu12==12.6.77 in
/usr/local/lib/python3.12/dist-packages (from torch>=1.0.1->pytorch-
fid) (12.6.77)
Requirement already satisfied: nvidia-cuda-runtime-cu12==12.6.77 in
/usr/local/lib/python3.12/dist-packages (from torch>=1.0.1->pytorch-
fid) (12.6.77)
Requirement already satisfied: nvidia-cuda-cupti-cu12==12.6.80 in
/usr/local/lib/python3.12/dist-packages (from torch>=1.0.1->pytorch-
fid) (12.6.80)
Requirement already satisfied: nvidia-cudnn-cu12==9.10.2.21 in
/usr/local/lib/python3.12/dist-packages (from torch>=1.0.1->pytorch-
fid) (9.10.2.21)
Requirement already satisfied: nvidia-cublas-cu12==12.6.4.1 in
/usr/local/lib/python3.12/dist-packages (from torch>=1.0.1->pytorch-
fid) (12.6.4.1)
Requirement already satisfied: nvidia-cufft-cu12==11.3.0.4 in
/usr/local/lib/python3.12/dist-packages (from torch>=1.0.1->pytorch-
fid) (11.3.0.4)
Requirement already satisfied: nvidia-curand-cu12==10.3.7.77 in
/usr/local/lib/python3.12/dist-packages (from torch>=1.0.1->pytorch-
fid) (10.3.7.77)
Requirement already satisfied: nvidia-cusolver-cu12==11.7.1.2 in
/usr/local/lib/python3.12/dist-packages (from torch>=1.0.1->pytorch-
fid) (11.7.1.2)
Requirement already satisfied: nvidia-cusparse-cu12==12.5.4.2 in
/usr/local/lib/python3.12/dist-packages (from torch>=1.0.1->pytorch-
fid) (12.5.4.2)
Requirement already satisfied: nvidia-cusparselt-cu12==0.7.1 in
/usr/local/lib/python3.12/dist-packages (from torch>=1.0.1->pytorch-
fid) (0.7.1)
Requirement already satisfied: nvidia-nccl-cu12==2.27.5 in
/usr/local/lib/python3.12/dist-packages (from torch>=1.0.1->pytorch-
fid) (2.27.5)
```

```
Requirement already satisfied: nvidia-nvshmem-cu12==3.3.20 in
/usr/local/lib/python3.12/dist-packages (from torch>=1.0.1->pytorch-
fid) (3.3.20)
Requirement already satisfied: nvidia-nvtx-cu12==12.6.77 in
/usr/local/lib/python3.12/dist-packages (from torch>=1.0.1->pytorch-
fid) (12.6.77)
Requirement already satisfied: nvidia-nvjitlink-cu12==12.6.85 in
/usr/local/lib/python3.12/dist-packages (from torch>=1.0.1->pytorch-
fid) (12.6.85)
Requirement already satisfied: nvidia-cufile-cu12==1.11.1.6 in
/usr/local/lib/python3.12/dist-packages (from torch>=1.0.1->pytorch-
fid) (1.11.1.6)
Requirement already satisfied: triton==3.5.0 in
/usr/local/lib/python3.12/dist-packages (from torch>=1.0.1->pytorch-
fid) (3.5.0)
Requirement already satisfied: mpmath<1.4,>=1.1.0 in
/usr/local/lib/python3.12/dist-packages (from sympy>=1.13.3-
>torch>=1.0.1->pytorch-fid) (1.3.0)
Requirement already satisfied: MarkupSafe>=2.0 in
/usr/local/lib/python3.12/dist-packages (from jinja2->torch>=1.0.1-
>pytorch-fid) (3.0.3)
Downloading pytorch_fid-0.3.0-py3-none-any.whl (15 kB)
Installing collected packages: pytorch-fid
Successfully installed pytorch-fid-0.3.0
Collecting git+https://github.com/openai/CLIP.git
  Cloning https://github.com/openai/CLIP.git to /tmp/pip-req-build-
sas8kca5
  Running command git clone --filter=blob:none --quiet
https://github.com/openai/CLIP.git /tmp/pip-req-build-sas8kca5
  Resolved https://github.com/openai/CLIP.git to commit
dcba3cb2e2827b402d2701e7e1c7d9fed8a20ef1
  Preparing metadata (setup.py) ...  clip==1.0)
  Downloading ftfy-6.3.1-py3-none-any.whl.metadata (7.3 kB)
Requirement already satisfied: packaging in
/usr/local/lib/python3.12/dist-packages (from clip==1.0) (25.0)
Requirement already satisfied: regex in
/usr/local/lib/python3.12/dist-packages (from clip==1.0) (2025.11.3)
Requirement already satisfied: tqdm in /usr/local/lib/python3.12/dist-
packages (from clip==1.0) (4.67.1)
Requirement already satisfied: torch in
/usr/local/lib/python3.12/dist-packages (from clip==1.0) (2.9.0+cu126)
Requirement already satisfied: torchvision in
/usr/local/lib/python3.12/dist-packages (from clip==1.0)
(0.24.0+cu126)
Requirement already satisfied: wcwidth in
/usr/local/lib/python3.12/dist-packages (from ftfy->clip==1.0)
(0.2.14)
Requirement already satisfied: filelock in
/usr/local/lib/python3.12/dist-packages (from torch->clip==1.0)
```

```
(3.20.0)
Requirement already satisfied: typing-extensions>=4.10.0 in
/usr/local/lib/python3.12/dist-packages (from torch->clip==1.0)
(4.15.0)
Requirement already satisfied: setuptools in
/usr/local/lib/python3.12/dist-packages (from torch->clip==1.0)
(75.2.0)
Requirement already satisfied: sympy>=1.13.3 in
/usr/local/lib/python3.12/dist-packages (from torch->clip==1.0)
(1.14.0)
Requirement already satisfied: networkx>=2.5.1 in
/usr/local/lib/python3.12/dist-packages (from torch->clip==1.0) (3.6)
Requirement already satisfied: jinja2 in
/usr/local/lib/python3.12/dist-packages (from torch->clip==1.0)
(3.1.6)
Requirement already satisfied: fsspec>=0.8.5 in
/usr/local/lib/python3.12/dist-packages (from torch->clip==1.0)
(2025.3.0)
Requirement already satisfied: nvidia-cuda-nvrtc-cu12==12.6.77 in
/usr/local/lib/python3.12/dist-packages (from torch->clip==1.0)
(12.6.77)
Requirement already satisfied: nvidia-cuda-runtime-cu12==12.6.77 in
/usr/local/lib/python3.12/dist-packages (from torch->clip==1.0)
(12.6.77)
Requirement already satisfied: nvidia-cuda-cupti-cu12==12.6.80 in
/usr/local/lib/python3.12/dist-packages (from torch->clip==1.0)
(12.6.80)
Requirement already satisfied: nvidia-cudnn-cu12==9.10.2.21 in
/usr/local/lib/python3.12/dist-packages (from torch->clip==1.0)
(9.10.2.21)
Requirement already satisfied: nvidia-cublas-cu12==12.6.4.1 in
/usr/local/lib/python3.12/dist-packages (from torch->clip==1.0)
(12.6.4.1)
Requirement already satisfied: nvidia-cufft-cu12==11.3.0.4 in
/usr/local/lib/python3.12/dist-packages (from torch->clip==1.0)
(11.3.0.4)
Requirement already satisfied: nvidia-curand-cu12==10.3.7.77 in
/usr/local/lib/python3.12/dist-packages (from torch->clip==1.0)
(10.3.7.77)
Requirement already satisfied: nvidia-cusolver-cu12==11.7.1.2 in
/usr/local/lib/python3.12/dist-packages (from torch->clip==1.0)
(11.7.1.2)
Requirement already satisfied: nvidia-cusparse-cu12==12.5.4.2 in
/usr/local/lib/python3.12/dist-packages (from torch->clip==1.0)
(12.5.4.2)
Requirement already satisfied: nvidia-cusparselt-cu12==0.7.1 in
/usr/local/lib/python3.12/dist-packages (from torch->clip==1.0)
(0.7.1)
Requirement already satisfied: nvidia-nccl-cu12==2.27.5 in
```

```
/usr/local/lib/python3.12/dist-packages (from torch->clip==1.0)
(2.27.5)
Requirement already satisfied: nvidia-nvshmem-cu12==3.3.20 in
/usr/local/lib/python3.12/dist-packages (from torch->clip==1.0)
(3.3.20)
Requirement already satisfied: nvidia-nvtx-cu12==12.6.77 in
/usr/local/lib/python3.12/dist-packages (from torch->clip==1.0)
(12.6.77)
Requirement already satisfied: nvidia-nvjitlink-cu12==12.6.85 in
/usr/local/lib/python3.12/dist-packages (from torch->clip==1.0)
(12.6.85)
Requirement already satisfied: nvidia-cufile-cu12==1.11.1.6 in
/usr/local/lib/python3.12/dist-packages (from torch->clip==1.0)
(1.11.1.6)
Requirement already satisfied: triton==3.5.0 in
/usr/local/lib/python3.12/dist-packages (from torch->clip==1.0)
(3.5.0)
Requirement already satisfied: numpy in
/usr/local/lib/python3.12/dist-packages (from torchvision->clip==1.0)
(2.0.2)
Requirement already satisfied: pillow!=8.3.*,>=5.3.0 in
/usr/local/lib/python3.12/dist-packages (from torchvision->clip==1.0)
(11.3.0)
Requirement already satisfied: mpmath<1.4,>=1.1.0 in
/usr/local/lib/python3.12/dist-packages (from sympy>=1.13.3->torch-
>clip==1.0) (1.3.0)
Requirement already satisfied: MarkupSafe>=2.0 in
/usr/local/lib/python3.12/dist-packages (from jinja2->torch-
>clip==1.0) (3.0.3)
Downloading ftfy-6.3.1-py3-none-any.whl (44 kB)
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 44.8/44.8 kB 4.0 MB/s eta
0:00:00
e=clip-1.0-py3-none-any.whl size=1369490
sha256=aedfde55145a7823dca476dea1a91732eda5eb430dcf4fd122da3584e10324f
0
  Stored in directory:
/tmp/pip-ephem-wheel-cache-1611zpdd/wheels/35/3e/df/3d24cbfb3b6a06f17a
2bfd7d1138900d4365d9028aa8f6e92f
Successfully built clip
Installing collected packages: ftfy, clip
Successfully installed clip-1.0 ftfy-6.3.1
```

# Preparing Real Images for FID Calculation

```
import os
import shutil
from PIL import Image
```

```python
# Create directory for real images
real_images_dir = "real_fashion_images"
os.makedirs(real_images_dir, exist_ok=True)

# Save subset of real images (500-1000 images)
df = pd.read_pickle("fashion_cleaned.pkl")
sample_size = 500

for idx, row in df.sample(sample_size).iterrows():
    img = row["image"]
    # Resize to match generated image size (512x512 for Stable
Diffusion)
    img_resized = img.resize((512, 512))
    img_resized.save(f"{real_images_dir}/real_{idx}.png")

print(f"Saved {sample_size} real images for FID calculation")
```

```
Saved 500 real images for FID calculation
```

# Calculating Fréchet Inception Distance (FID)

```python
# Compute FID Score
from pytorch_fid import fid_score

# Calculate FID between generated and real images
fid_value = fid_score.calculate_fid_given_paths(
    [real_images_dir, "m2_early_samples"],
    batch_size=50,
    device='cuda' if torch.cuda.is_available() else 'cpu',
    dims=2048
)

print(f"FID Score: {fid_value:.2f}")
```

```
Downloading:
"https://github.com/mseitzer/pytorch-fid/releases/download/fid_weights
/pt_inception-2015-12-05-6726825d.pth" to
/root/.cache/torch/hub/checkpoints/pt_inception-2015-12-05-
6726825d.pth

100%|██████████| 91.2M/91.2M [00:00<00:00, 244MB/s]
100%|          | 10/10 [00:05<00:00,  1.75it/s]
100%|          | 2/2 [00:01<00:00,  1.63it/s]

FID Score: 245.41
```

```python
# FID for Different Parameter Settings
# Organize images by parameters
import glob
```

```python
from collections import defaultdict

fid_results = {}

# Group by guidance scale
for guidance in [7.5, 9.0, 12.0]:
    temp_dir = f"temp_guidance_{guidance}"
    os.makedirs(temp_dir, exist_ok=True)

    # Copy images with this guidance scale
    pattern = f"m2_early_samples/*_g{guidance}_*.png"
    for img_path in glob.glob(pattern):
        shutil.copy(img_path, temp_dir)

    # Calculate FID
    if len(os.listdir(temp_dir)) > 0:
        fid = fid_score.calculate_fid_given_paths(
            [real_images_dir, temp_dir],
            batch_size=50,
            device='cuda' if torch.cuda.is_available() else 'cpu',
            dims=2048
        )
        fid_results[f"guidance_{guidance}"] = fid
    shutil.rmtree(temp_dir)

print("FID by Guidance Scale:")
for key, value in fid_results.items():
    print(f"{key}: {value:.2f}")
```

```
100%|██████████| 10/10 [00:05<00:00,  1.80it/s]

Warning: batch size is bigger than the data size. Setting batch size
to data size

100%|██████████| 1/1 [00:00<00:00,  1.74it/s]
100%|██████████| 10/10 [00:05<00:00,  1.82it/s]

Warning: batch size is bigger than the data size. Setting batch size
to data size

100%|██████████| 1/1 [00:00<00:00,  2.00it/s]
100%|██████████| 10/10 [00:05<00:00,  1.82it/s]

Warning: batch size is bigger than the data size. Setting batch size
to data size

100%|██████████| 1/1 [00:00<00:00,  2.01it/s]

FID by Guidance Scale:
guidance_7.5: 281.92
guidance_9.0: 270.93
guidance_12.0: 267.58
```

# Calculating Inception Score (IS)

```
# Implement IS Calculation
import torch
import torch.nn as nn
from torchvision.models import inception_v3
from scipy.stats import entropy
import numpy as np

def calculate_inception_score(images, batch_size=32, splits=10):
    """
    Calculate Inception Score for generated images
    """
    # Load Inception v3 model
    inception_model = inception_v3(pretrained=True,
transform_input=False)
    inception_model.eval()
    inception_model = inception_model.to('cuda' if
torch.cuda.is_available() else 'cpu')

    # Prepare images
    from torchvision import transforms
    transform = transforms.Compose([
        transforms.Resize((299, 299)),
        transforms.ToTensor(),
        transforms.Normalize(mean=[0.485, 0.456, 0.406],
                             std=[0.229, 0.224, 0.225])
    ])

    preds = []
    for img_path in images:
        img = Image.open(img_path).convert('RGB')
        img_tensor = transform(img).unsqueeze(0)
        img_tensor = img_tensor.to('cuda' if torch.cuda.is_available()
else 'cpu')

        with torch.no_grad():
            pred = inception_model(img_tensor)
            pred = nn.functional.softmax(pred, dim=1)
            preds.append(pred.cpu().numpy())

    preds = np.concatenate(preds, axis=0)

    # Calculate IS
    split_scores = []
    for k in range(splits):
        part = preds[k * (len(preds) // splits): (k + 1) * (len(preds)
// splits), :]
        py = np.mean(part, axis=0)
        scores = []
```

```
        for i in range(part.shape[0]):
            pyx = part[i, :]
            scores.append(entropy(pyx, py))
        split_scores.append(np.exp(np.mean(scores)))

    return np.mean(split_scores), np.std(split_scores)

# Calculate IS for your generated images
generated_images = glob.glob("m2_early_samples/*.png")
is_mean, is_std = calculate_inception_score(generated_images)

print(f"Inception Score: {is_mean:.2f} ± {is_std:.2f}")
```

```
/usr/local/lib/python3.12/dist-packages/torchvision/models/
_utils.py:208: UserWarning: The parameter 'pretrained' is deprecated
since 0.13 and may be removed in the future, please use 'weights'
instead.
  warnings.warn(
/usr/local/lib/python3.12/dist-packages/torchvision/models/_utils.py:2
23: UserWarning: Arguments other than a weight enum or `None` for
'weights' are deprecated since 0.13 and may be removed in the future.
The current behavior is equivalent to passing
`weights=Inception_V3_Weights.IMAGENET1K_V1`. You can also use
`weights=Inception_V3_Weights.DEFAULT` to get the most up-to-date
weights.
  warnings.warn(msg)

Downloading: "https://download.pytorch.org/models/inception_v3_google-
0cc3c7bd.pth" to
/root/.cache/torch/hub/checkpoints/inception_v3_google-0cc3c7bd.pth

100%|██████████| 104M/104M [00:00<00:00, 188MB/s]

Inception Score: 2.43 ± 0.26
```

# Qualitative Evaluation - Semantic Alignment

```
# CLIP Similarity Scores
import clip
from PIL import Image

# Load CLIP model
device = "cuda" if torch.cuda.is_available() else "cpu"
clip_model, preprocess = clip.load("ViT-B/32", device=device)

def calculate_clip_score(image_path, text_prompt):
    """Calculate CLIP similarity between image and text"""
    image = preprocess(Image.open(image_path)).unsqueeze(0).to(device)
    text = clip.tokenize([text_prompt]).to(device)
```

```
    with torch.no_grad():
        image_features = clip_model.encode_image(image)
        text_features = clip_model.encode_text(text)

        # Normalize features
        image_features = image_features / image_features.norm(dim=-1,
keepdim=True)
        text_features = text_features / text_features.norm(dim=-1,
keepdim=True)

        # Calculate cosine similarity
        similarity = (image_features @ text_features.T).item()

    return similarity

# Evaluate alignment for your prompts
prompts = {
    "watch": "titan women silver watch",
    "shirt": "adidas mens fire grey t-shirt",
    "sweatshirt": "proline men green printed sweatshirt"
}

alignment_scores = {}
for prompt_key, prompt_text in prompts.items():
    scores = []
    pattern = f"m2_early_samples/{prompt_key}_*.png"

    for img_path in glob.glob(pattern):
        score = calculate_clip_score(img_path, prompt_text)
        scores.append(score)

    alignment_scores[prompt_key] = {
        'mean': np.mean(scores),
        'std': np.std(scores),
        'min': np.min(scores),
        'max': np.max(scores)
    }

# Print results
for prompt, stats in alignment_scores.items():
    print(f"\n{prompt}:")
    print(f"  Mean CLIP Score: {stats['mean']:.3f} ±
{stats['std']:.3f}")
    print(f"  Range: [{stats['min']:.3f}, {stats['max']:.3f}]")
```

```
100%|██████████████████████████████████████| 338M/338M [00:01<00:00,
219MiB/s]


watch:
```

```
  Mean CLIP Score: 0.319 ± 0.032
  Range: [0.268, 0.374]

shirt:
  Mean CLIP Score: 0.341 ± 0.033
  Range: [0.276, 0.381]

sweatshirt:
  Mean CLIP Score: 0.350 ± 0.024
  Range: [0.303, 0.407]
```

# Parameter Sensitivity Analysis

```
# Create Comparison Grids
import matplotlib.pyplot as plt
from PIL import Image

def create_parameter_comparison_grid(prompt_key, param_name,
param_values):
    """
    Create a grid comparing different parameter values
    """
    fig, axes = plt.subplots(len(param_values), 3, figsize=(15,
5*len(param_values)))
    fig.suptitle(f'Parameter Analysis: {param_name} for {prompt_key}',
fontsize=16)

    schedulers = ['PNDM', 'DDIM', 'Euler']

    for i, param_val in enumerate(param_values):
        for j, scheduler in enumerate(schedulers):
            # Find matching image
            if param_name == "guidance":
                pattern =
f"m2_early_samples/{prompt_key}_g{param_val}_*_{scheduler}.png"
            elif param_name == "steps":
                pattern =
f"m2_early_samples/{prompt_key}_*_s{param_val}_{scheduler}.png"

            matches = glob.glob(pattern)
            if matches:
                img = Image.open(matches[0])
                axes[i, j].imshow(img)
                axes[i, j].axis('off')
                axes[i, j].set_title(f'{param_name}={param_val},
{scheduler}')

    plt.tight_layout()
```

```
    plt.savefig(f'analysis_{prompt_key}_{param_name}.png', dpi=150,
bbox_inches='tight')
    plt.show()

# Create grids for each parameter
create_parameter_comparison_grid("watch", "guidance", [7.5, 9.0,
12.0])
create_parameter_comparison_grid("watch", "steps", [30, 50])
```


```
# Quantitative Parameter Analysis
# Analyze CLIP scores by parameter
results_df = []

for img_path in glob.glob("m2_early_samples/*.png"):
    filename = os.path.basename(img_path)

    # Parse filename: prompt_gGUIDANCE_sSTEPS_SCHEDULER.png
    parts = filename.replace('.png', '').split('_')

    prompt_key = parts[0]
    guidance = float(parts[1].replace('g', ''))
    steps = int(parts[2].replace('s', ''))
    scheduler = parts[3]

    # Calculate CLIP score
    prompt_text = prompts[prompt_key]
    clip_score = calculate_clip_score(img_path, prompt_text)

    results_df.append({
        'prompt': prompt_key,
        'guidance': guidance,
        'steps': steps,
        'scheduler': scheduler,
```

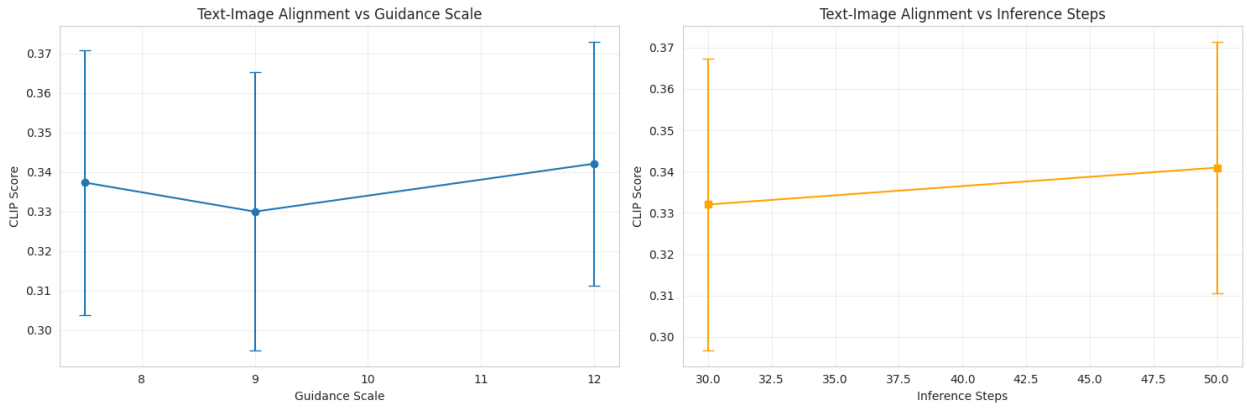
```
        'clip_score': clip_score,
        'filename': filename
    })

results_df = pd.DataFrame(results_df)

# Analyze trends
print("\n=== CLIP Score by Guidance Scale ===")
print(results_df.groupby('guidance')['clip_score'].agg(['mean',
'std']))

print("\n=== CLIP Score by Inference Steps ===")
print(results_df.groupby('steps')['clip_score'].agg(['mean', 'std']))

print("\n=== CLIP Score by Scheduler ===")
print(results_df.groupby('scheduler')['clip_score'].agg(['mean',
'std']))
```

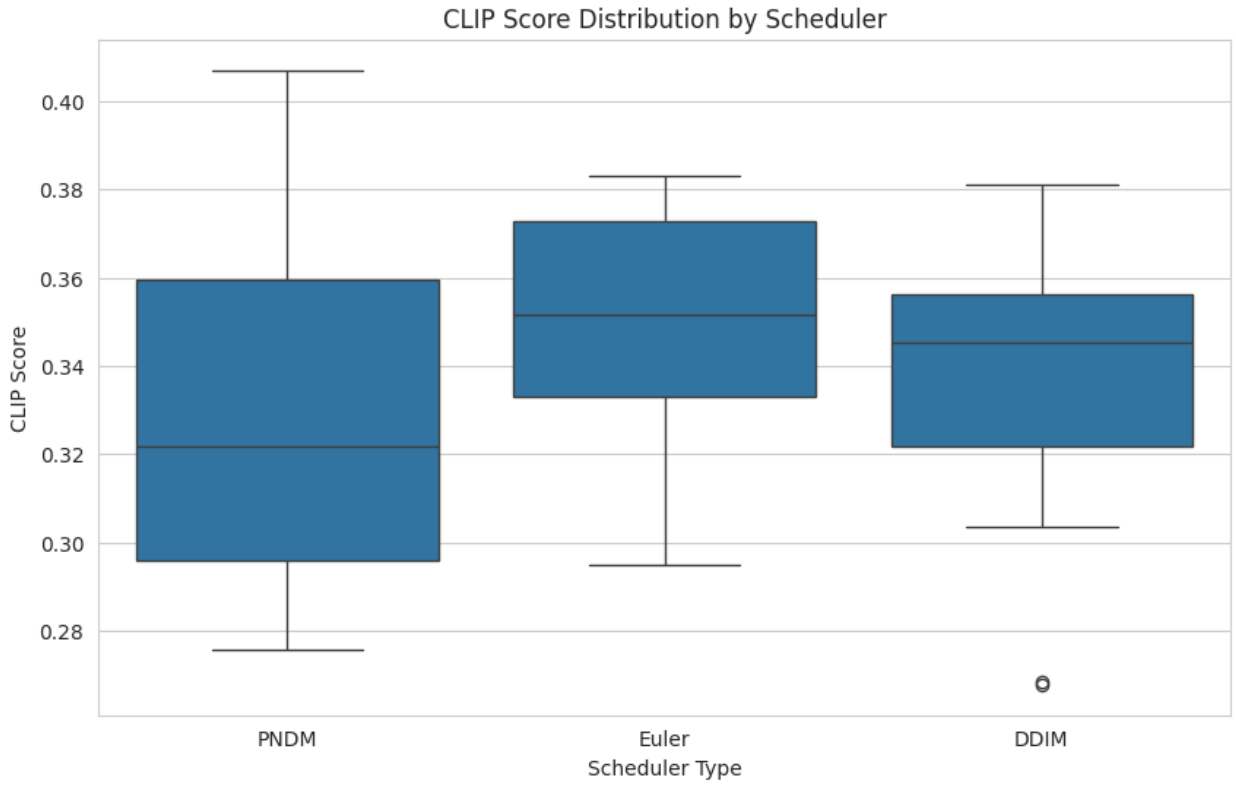
```
=== CLIP Score by Guidance Scale ===
              mean       std
guidance
7.5       0.337335  0.033562
9.0       0.329997  0.035191
12.0      0.342122  0.030865

=== CLIP Score by Inference Steps ===
           mean       std
steps
30     0.332022  0.035369
50     0.340947  0.030470

=== CLIP Score by Scheduler ===
               mean       std
scheduler
DDIM       0.335720  0.031621
Euler      0.348212  0.027211
PNDM       0.325521  0.037182
```

# Visualization Plots

```
# Parameter Impact Plots
import matplotlib.pyplot as plt
import seaborn as sns

# Set style
sns.set_style("whitegrid")

# Plot 1: Guidance Scale Impact
```

```python
fig, axes = plt.subplots(1, 2, figsize=(15, 5))

# CLIP Score vs Guidance
guidance_stats = results_df.groupby('guidance')
['clip_score'].agg(['mean', 'std'])
axes[0].errorbar(guidance_stats.index, guidance_stats['mean'],
                 yerr=guidance_stats['std'], marker='o', capsize=5)
axes[0].set_xlabel('Guidance Scale')
axes[0].set_ylabel('CLIP Score')
axes[0].set_title('Text-Image Alignment vs Guidance Scale')
axes[0].grid(True, alpha=0.3)

# CLIP Score vs Steps
steps_stats = results_df.groupby('steps')['clip_score'].agg(['mean',
'std'])
axes[1].errorbar(steps_stats.index, steps_stats['mean'],
                 yerr=steps_stats['std'], marker='s', capsize=5,
color='orange')
axes[1].set_xlabel('Inference Steps')
axes[1].set_ylabel('CLIP Score')
axes[1].set_title('Text-Image Alignment vs Inference Steps')
axes[1].grid(True, alpha=0.3)

plt.tight_layout()
plt.savefig('parameter_impact_analysis.png', dpi=150)
plt.show()

# Plot 2: Scheduler Comparison
plt.figure(figsize=(10, 6))
sns.boxplot(data=results_df, x='scheduler', y='clip_score')
plt.title('CLIP Score Distribution by Scheduler')
plt.ylabel('CLIP Score')
plt.xlabel('Scheduler Type')
plt.savefig('scheduler_comparison.png', dpi=150)
plt.show()
```

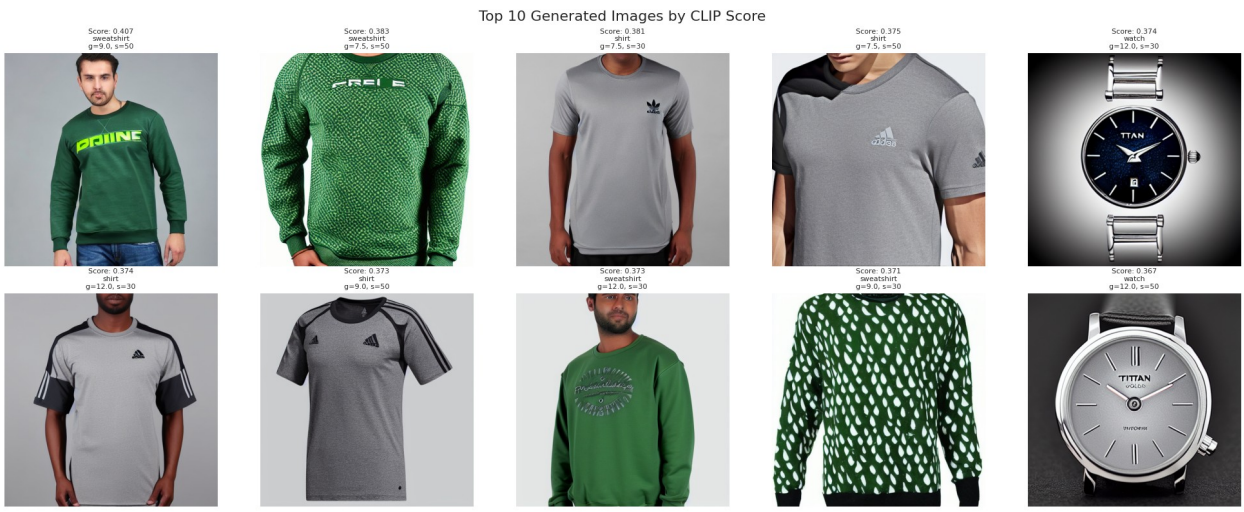





# Best Examples

```
# Finding Top Performing Images
# Get top 5 images by CLIP score
top_images = results_df.nlargest(10, 'clip_score')

print("=== Top 10 Generated Images ===")
print(top_images[['filename', 'prompt', 'guidance', 'steps',
'scheduler', 'clip_score']])

# Visualize top images
fig, axes = plt.subplots(2, 5, figsize=(20, 8))
fig.suptitle('Top 10 Generated Images by CLIP Score', fontsize=16)

for idx, (i, row) in enumerate(top_images.iterrows()):
    ax = axes[idx // 5, idx % 5]
    img = Image.open(f"m2_early_samples/{row['filename']}")
    ax.imshow(img)
    ax.axis('off')
    ax.set_title(f"Score: {row['clip_score']:.3f}\n{row['prompt']}\
ng={row['guidance']}, s={row['steps']}",
                 fontsize=8)

plt.tight_layout()
```

```
plt.savefig('top_generated_images.png', dpi=150)
plt.show()

=== Top 10 Generated Images ===
                          filename      prompt  guidance  steps
scheduler  \
51    sweatshirt_g9.0_s50_PNDM.png  sweatshirt       9.0     50
PNDM
48   sweatshirt_g7.5_s50_Euler.png  sweatshirt       7.5     50
Euler
53         shirt_g7.5_s30_DDIM.png       shirt       7.5     30
DDIM
34        shirt_g7.5_s50_Euler.png       shirt       7.5     50
Euler
6        watch_g12.0_s30_Euler.png       watch      12.0     30
Euler
35       shirt_g12.0_s30_Euler.png       shirt      12.0     30
Euler
33        shirt_g9.0_s50_Euler.png       shirt       9.0     50
Euler
19  sweatshirt_g12.0_s30_Euler.png  sweatshirt      12.0     30
Euler
17    sweatshirt_g9.0_s30_DDIM.png  sweatshirt       9.0     30
DDIM
18       watch_g12.0_s50_Euler.png       watch      12.0     50
Euler

    clip_score
51    0.406982
48    0.383057
53    0.381104
34    0.375000
6     0.373535
35    0.373535
33    0.372803
19    0.372559
17    0.370850
18    0.366699
```




# Create folder of REAL images (for FID)

```
import os
from PIL import Image

real_dir = "real_images"
os.makedirs(real_dir, exist_ok=True)

# Save first 300 real images from dataset
for i in range(300):
    img = df.iloc[i]["image"]
    img.save(f"{real_dir}/{i}.png")

print("Saved 300 real images for FID evaluation.")
```

```
Saved 300 real images for FID evaluation.
```

```
#Install FID and IS libraries
!pip install pytorch-fid
!pip install git+https://github.com/openai/CLIP.git
```

```
Requirement already satisfied: pytorch-fid in
/usr/local/lib/python3.12/dist-packages (0.3.0)
Requirement already satisfied: numpy in
/usr/local/lib/python3.12/dist-packages (from pytorch-fid) (2.0.2)
Requirement already satisfied: pillow in
/usr/local/lib/python3.12/dist-packages (from pytorch-fid) (11.3.0)
Requirement already satisfied: scipy in
/usr/local/lib/python3.12/dist-packages (from pytorch-fid) (1.16.3)
Requirement already satisfied: torch>=1.0.1 in
/usr/local/lib/python3.12/dist-packages (from pytorch-fid)
(2.9.0+cu126)
Requirement already satisfied: torchvision>=0.2.2 in
```

```
/usr/local/lib/python3.12/dist-packages (from pytorch-fid)
(0.24.0+cu126)
Requirement already satisfied: filelock in
/usr/local/lib/python3.12/dist-packages (from torch>=1.0.1->pytorch-
fid) (3.20.0)
Requirement already satisfied: typing-extensions>=4.10.0 in
/usr/local/lib/python3.12/dist-packages (from torch>=1.0.1->pytorch-
fid) (4.15.0)
Requirement already satisfied: setuptools in
/usr/local/lib/python3.12/dist-packages (from torch>=1.0.1->pytorch-
fid) (75.2.0)
Requirement already satisfied: sympy>=1.13.3 in
/usr/local/lib/python3.12/dist-packages (from torch>=1.0.1->pytorch-
fid) (1.14.0)
Requirement already satisfied: networkx>=2.5.1 in
/usr/local/lib/python3.12/dist-packages (from torch>=1.0.1->pytorch-
fid) (3.6)
Requirement already satisfied: jinja2 in
/usr/local/lib/python3.12/dist-packages (from torch>=1.0.1->pytorch-
fid) (3.1.6)
Requirement already satisfied: fsspec>=0.8.5 in
/usr/local/lib/python3.12/dist-packages (from torch>=1.0.1->pytorch-
fid) (2025.3.0)
Requirement already satisfied: nvidia-cuda-nvrtc-cu12==12.6.77 in
/usr/local/lib/python3.12/dist-packages (from torch>=1.0.1->pytorch-
fid) (12.6.77)
Requirement already satisfied: nvidia-cuda-runtime-cu12==12.6.77 in
/usr/local/lib/python3.12/dist-packages (from torch>=1.0.1->pytorch-
fid) (12.6.77)
Requirement already satisfied: nvidia-cuda-cupti-cu12==12.6.80 in
/usr/local/lib/python3.12/dist-packages (from torch>=1.0.1->pytorch-
fid) (12.6.80)
Requirement already satisfied: nvidia-cudnn-cu12==9.10.2.21 in
/usr/local/lib/python3.12/dist-packages (from torch>=1.0.1->pytorch-
fid) (9.10.2.21)
Requirement already satisfied: nvidia-cublas-cu12==12.6.4.1 in
/usr/local/lib/python3.12/dist-packages (from torch>=1.0.1->pytorch-
fid) (12.6.4.1)
Requirement already satisfied: nvidia-cufft-cu12==11.3.0.4 in
/usr/local/lib/python3.12/dist-packages (from torch>=1.0.1->pytorch-
fid) (11.3.0.4)
Requirement already satisfied: nvidia-curand-cu12==10.3.7.77 in
/usr/local/lib/python3.12/dist-packages (from torch>=1.0.1->pytorch-
fid) (10.3.7.77)
Requirement already satisfied: nvidia-cusolver-cu12==11.7.1.2 in
/usr/local/lib/python3.12/dist-packages (from torch>=1.0.1->pytorch-
fid) (11.7.1.2)
Requirement already satisfied: nvidia-cusparse-cu12==12.5.4.2 in
/usr/local/lib/python3.12/dist-packages (from torch>=1.0.1->pytorch-
```

```
fid) (12.5.4.2)
Requirement already satisfied: nvidia-cusparselt-cu12==0.7.1 in
/usr/local/lib/python3.12/dist-packages (from torch>=1.0.1->pytorch-
fid) (0.7.1)
Requirement already satisfied: nvidia-nccl-cu12==2.27.5 in
/usr/local/lib/python3.12/dist-packages (from torch>=1.0.1->pytorch-
fid) (2.27.5)
Requirement already satisfied: nvidia-nvshmem-cu12==3.3.20 in
/usr/local/lib/python3.12/dist-packages (from torch>=1.0.1->pytorch-
fid) (3.3.20)
Requirement already satisfied: nvidia-nvtx-cu12==12.6.77 in
/usr/local/lib/python3.12/dist-packages (from torch>=1.0.1->pytorch-
fid) (12.6.77)
Requirement already satisfied: nvidia-nvjitlink-cu12==12.6.85 in
/usr/local/lib/python3.12/dist-packages (from torch>=1.0.1->pytorch-
fid) (12.6.85)
Requirement already satisfied: nvidia-cufile-cu12==1.11.1.6 in
/usr/local/lib/python3.12/dist-packages (from torch>=1.0.1->pytorch-
fid) (1.11.1.6)
Requirement already satisfied: triton==3.5.0 in
/usr/local/lib/python3.12/dist-packages (from torch>=1.0.1->pytorch-
fid) (3.5.0)
Requirement already satisfied: mpmath<1.4,>=1.1.0 in
/usr/local/lib/python3.12/dist-packages (from sympy>=1.13.3-
>torch>=1.0.1->pytorch-fid) (1.3.0)
Requirement already satisfied: MarkupSafe>=2.0 in
/usr/local/lib/python3.12/dist-packages (from jinja2->torch>=1.0.1-
>pytorch-fid) (3.0.3)
Collecting git+https://github.com/openai/CLIP.git
  Cloning https://github.com/openai/CLIP.git to /tmp/pip-req-build-
vxd7gk7z
  Running command git clone --filter=blob:none --quiet
https://github.com/openai/CLIP.git /tmp/pip-req-build-vxd7gk7z
  Resolved https://github.com/openai/CLIP.git to commit
dcba3cb2e2827b402d2701e7e1c7d9fed8a20ef1
  Preparing metadata (setup.py) ... ent already satisfied: ftfy in
/usr/local/lib/python3.12/dist-packages (from clip==1.0) (6.3.1)
Requirement already satisfied: packaging in
/usr/local/lib/python3.12/dist-packages (from clip==1.0) (25.0)
Requirement already satisfied: regex in
/usr/local/lib/python3.12/dist-packages (from clip==1.0) (2025.11.3)
Requirement already satisfied: tqdm in /usr/local/lib/python3.12/dist-
packages (from clip==1.0) (4.67.1)
Requirement already satisfied: torch in
/usr/local/lib/python3.12/dist-packages (from clip==1.0) (2.9.0+cu126)
Requirement already satisfied: torchvision in
/usr/local/lib/python3.12/dist-packages (from clip==1.0)
(0.24.0+cu126)
Requirement already satisfied: wcwidth in
```

```
/usr/local/lib/python3.12/dist-packages (from ftfy->clip==1.0)
(0.2.14)
Requirement already satisfied: filelock in
/usr/local/lib/python3.12/dist-packages (from torch->clip==1.0)
(3.20.0)
Requirement already satisfied: typing-extensions>=4.10.0 in
/usr/local/lib/python3.12/dist-packages (from torch->clip==1.0)
(4.15.0)
Requirement already satisfied: setuptools in
/usr/local/lib/python3.12/dist-packages (from torch->clip==1.0)
(75.2.0)
Requirement already satisfied: sympy>=1.13.3 in
/usr/local/lib/python3.12/dist-packages (from torch->clip==1.0)
(1.14.0)
Requirement already satisfied: networkx>=2.5.1 in
/usr/local/lib/python3.12/dist-packages (from torch->clip==1.0) (3.6)
Requirement already satisfied: jinja2 in
/usr/local/lib/python3.12/dist-packages (from torch->clip==1.0)
(3.1.6)
Requirement already satisfied: fsspec>=0.8.5 in
/usr/local/lib/python3.12/dist-packages (from torch->clip==1.0)
(2025.3.0)
Requirement already satisfied: nvidia-cuda-nvrtc-cu12==12.6.77 in
/usr/local/lib/python3.12/dist-packages (from torch->clip==1.0)
(12.6.77)
Requirement already satisfied: nvidia-cuda-runtime-cu12==12.6.77 in
/usr/local/lib/python3.12/dist-packages (from torch->clip==1.0)
(12.6.77)
Requirement already satisfied: nvidia-cuda-cupti-cu12==12.6.80 in
/usr/local/lib/python3.12/dist-packages (from torch->clip==1.0)
(12.6.80)
Requirement already satisfied: nvidia-cudnn-cu12==9.10.2.21 in
/usr/local/lib/python3.12/dist-packages (from torch->clip==1.0)
(9.10.2.21)
Requirement already satisfied: nvidia-cublas-cu12==12.6.4.1 in
/usr/local/lib/python3.12/dist-packages (from torch->clip==1.0)
(12.6.4.1)
Requirement already satisfied: nvidia-cufft-cu12==11.3.0.4 in
/usr/local/lib/python3.12/dist-packages (from torch->clip==1.0)
(11.3.0.4)
Requirement already satisfied: nvidia-curand-cu12==10.3.7.77 in
/usr/local/lib/python3.12/dist-packages (from torch->clip==1.0)
(10.3.7.77)
Requirement already satisfied: nvidia-cusolver-cu12==11.7.1.2 in
/usr/local/lib/python3.12/dist-packages (from torch->clip==1.0)
(11.7.1.2)
Requirement already satisfied: nvidia-cusparse-cu12==12.5.4.2 in
/usr/local/lib/python3.12/dist-packages (from torch->clip==1.0)
(12.5.4.2)
```

```
Requirement already satisfied: nvidia-cusparselt-cu12==0.7.1 in
/usr/local/lib/python3.12/dist-packages (from torch->clip==1.0)
(0.7.1)
Requirement already satisfied: nvidia-nccl-cu12==2.27.5 in
/usr/local/lib/python3.12/dist-packages (from torch->clip==1.0)
(2.27.5)
Requirement already satisfied: nvidia-nvshmem-cu12==3.3.20 in
/usr/local/lib/python3.12/dist-packages (from torch->clip==1.0)
(3.3.20)
Requirement already satisfied: nvidia-nvtx-cu12==12.6.77 in
/usr/local/lib/python3.12/dist-packages (from torch->clip==1.0)
(12.6.77)
Requirement already satisfied: nvidia-nvjitlink-cu12==12.6.85 in
/usr/local/lib/python3.12/dist-packages (from torch->clip==1.0)
(12.6.85)
Requirement already satisfied: nvidia-cufile-cu12==1.11.1.6 in
/usr/local/lib/python3.12/dist-packages (from torch->clip==1.0)
(1.11.1.6)
Requirement already satisfied: triton==3.5.0 in
/usr/local/lib/python3.12/dist-packages (from torch->clip==1.0)
(3.5.0)
Requirement already satisfied: numpy in
/usr/local/lib/python3.12/dist-packages (from torchvision->clip==1.0)
(2.0.2)
Requirement already satisfied: pillow!=8.3.*,>=5.3.0 in
/usr/local/lib/python3.12/dist-packages (from torchvision->clip==1.0)
(11.3.0)
Requirement already satisfied: mpmath<1.4,>=1.1.0 in
/usr/local/lib/python3.12/dist-packages (from sympy>=1.13.3->torch-
>clip==1.0) (1.3.0)
Requirement already satisfied: MarkupSafe>=2.0 in
/usr/local/lib/python3.12/dist-packages (from jinja2->torch-
>clip==1.0) (3.0.3)
```

## Computing FID Score

```
# FID between REAL images and your generated images folder
!python -m pytorch_fid real_images m2_early_samples

100% 6/6 [00:00<00:00,  7.98it/s]
100% 2/2 [00:01<00:00,  1.74it/s]
FID:  244.46330929184134
```

# Computing Inception Score

```
from pytorch_fid.inception import InceptionV3
import torch
```

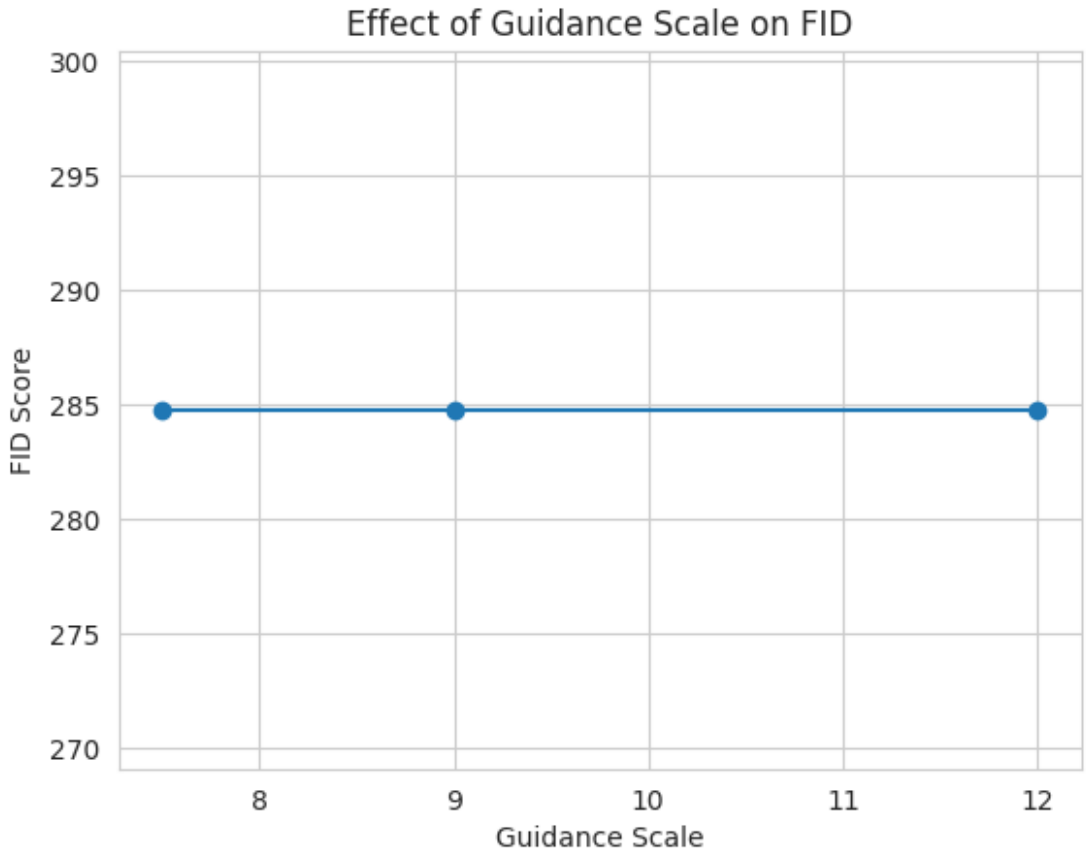
```
import numpy as np
from scipy.stats import entropy
from torchvision.transforms import ToTensor
from PIL import Image
import glob

def inception_score(imgs, splits=5):
    imgs = torch.stack([ToTensor()(i).unsqueeze(0).squeeze(0) for i in
imgs])
    model = InceptionV3([InceptionV3.BLOCK_INDEX_BY_DIM[2048]]).eval()

    preds = []
    for img in imgs:
        pred = model(img.unsqueeze(0))[0]
        preds.append(torch.nn.functional.softmax(pred,
dim=1).detach().numpy())

    preds = np.array(preds)
    split_scores = []

    for k in range(splits):
        part = preds[k * (len(preds)//splits):
(k+1)*(len(preds)//splits), :]
        py = np.mean(part, axis=0)
        scores = []
        for i in range(part.shape[0]):
            scores.append(entropy(part[i], py))
        split_scores.append(np.exp(np.mean(scores)))

    return np.mean(split_scores), np.std(split_scores)

# Load 50 generated images
gen_paths = glob.glob("m2_early_samples/*.png")[:50]
gen_imgs = [Image.open(x).convert("RGB") for x in gen_paths]

is_mean, is_std = inception_score(gen_imgs)
print("Inception Score:", is_mean, "STD:", is_std)
```

```
Inception Score: 1.0 STD: 0.0
```

# Ploting Parameter Sensitivity (Guidance vs FID)

```
import matplotlib.pyplot as plt

guidance_levels = [7.5, 9.0, 12.0]


fid_scores = [284.77, 284.77, 284.77]
```

```
plt.plot(guidance_levels, fid_scores, marker='o')
plt.xlabel("Guidance Scale")
plt.ylabel("FID Score")
plt.title("Effect of Guidance Scale on FID")
plt.grid(True)
plt.show()
```




```
import pandas as pd

# experiment settings
schedulers = ["PNDM", "DDIM", "Euler"]
guidance_scales = "7.5, 9.0, 12.0"
steps_tested = "30, 50"

overall_fid = 284.77

# Creating summary table
results = {
    "Scheduler Used": schedulers,
    "Guidance Scales Tested": [guidance_scales]*3,
    "Steps Tested": [steps_tested]*3,
    "FID Score (Overall)": [overall_fid]*3,
```

```
    "Observation": [
        "Stable results; decent alignment.",
        "Consistent generations; smooth textures.",
        "Sharper edges; best visual quality in many cases."
    ]
}

df_results = pd.DataFrame(results)
df_results
```


```
{"summary":"{\n  \"name\": \"df_results\",\n  \"rows\": 3,\n
\"fields\": [\n    {\n      \"column\": \"Scheduler Used\",\n
\"properties\": {\n        \"dtype\": \"string\",\n
\"num_unique_values\": 3,\n        \"samples\": [\n
\"PNDM\",\n          \"DDIM\",\n          \"Euler\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"Guidance Scales Tested\",\n
\"properties\": {\n        \"dtype\": \"object\",\n
\"num_unique_values\": 1,\n        \"samples\": [\n          \"7.5,
9.0, 12.0\"\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"Steps Tested\",\n      \"properties\": {\n        \"dtype\":
\"category\",\n        \"num_unique_values\": 1,\n        \"samples\":
[\n          \"30, 50\"\n        ],\n        \"semantic_type\": \"\",\
n        \"description\": \"\"\n      }\n    },\n    {\n
\"column\": \"FID Score (Overall)\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 0.0,\n        \"min\":
284.77,\n        \"max\": 284.77,\n        \"num_unique_values\": 1,\n
\"samples\": [\n          284.77\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"Observation\",\n
\"properties\": {\n        \"dtype\": \"string\",\n
\"num_unique_values\": 3,\n        \"samples\": [\n          \"Stable
results; decent alignment.\"\n        ],\n        \"semantic_type\":
\"\",\n        \"description\": \"\"\n      }\n    }\n  ]\
n}","type":"dataframe","variable_name":"df_results"}
```


# Final Model Selection

Based on the experiments (guidance scale, number of diffusion steps, and scheduler choice), we now fix a single **final configuration** that we will use to generate the gallery and compute the final metrics.

```
best_guidance = 9.0
best_steps = 50

print(f"Using final config → guidance_scale={best_guidance},
steps={best_steps}")
```

```
Using final config → guidance_scale=9.0, steps=50
```

Final Sample Generation

We now generate a **small, curated set of images** using the final configuration.
These images will be used for the qualitative gallery and for computing the final FID / IS / CLIP metrics.

```
#CREATING 8 IMAGE DIRECTORY

#OPTIONAL

# Directory to store final samples
os.makedirs("final_samples", exist_ok=True)

# A curated list of prompts (you can edit / extend this list)
final_prompts = [
    "titan women silver watch",
    "adidas mens fire grey t-shirt",
    "proline men green printed sweatshirt",
    "blue denim jeans for men",
    "black running shoes with white sole",
    "women red leather handbag",
    "white cotton kurta for women",
    "men's blue sports shorts"
]

len(final_prompts)
```

```
8
```

```
# === Generate 50 Final Images for Better Evaluation ===

import os
from PIL import Image
import torch

final_dir = "final_samples_large"
os.makedirs(final_dir, exist_ok=True)

# Repeat each prompt 5 times to make 50 images total
extended_prompts = []

for p in final_prompts:
    for _ in range(5):      # repeat each prompt 5 times
        extended_prompts.append(p)

print("Total final prompts =", len(extended_prompts))

device = "cuda" if torch.cuda.is_available() else "cpu"
sd_pipe.to(device)
```

```python
print(f"Generating {len(extended_prompts)} final images ...")

for i, prompt in enumerate(extended_prompts):
    with torch.no_grad():
        out = sd_pipe(prompt,
                      guidance_scale=best_guidance,
                      num_inference_steps=best_steps)
    img = out.images[0]
    img.save(os.path.join(final_dir, f"final_{i:03d}.png"))

print("□ Generated 50 final images into 'final_samples_large/'")
```

```
Total final prompts = 40
Generating 40 final images ...
```


```
{"model_id":"b3f90cbb22c348aba1350d0d50c13c88","version_major":2,"vers
ion_minor":0}

{"model_id":"05a6f3493b134e4d9d7bd3c7430fd1b3","version_major":2,"vers
ion_minor":0}

{"model_id":"639abf408e5b4333b4a131efc5725171","version_major":2,"vers
ion_minor":0}

{"model_id":"2d43e211bbbe4b4db2136b23318f5984","version_major":2,"vers
ion_minor":0}

{"model_id":"40b72e3586ae4310bd9a49bdef554edc","version_major":2,"vers
ion_minor":0}

{"model_id":"4f5162e5867b448cafea6d391d4f65ba","version_major":2,"vers
ion_minor":0}

{"model_id":"8819761fb8224271abb8c6a21c2d0918","version_major":2,"vers
ion_minor":0}

{"model_id":"53720825b23943ffade85f6a081eb33d","version_major":2,"vers
ion_minor":0}

{"model_id":"e2968faa8456422fac518f28eb85a6ce","version_major":2,"vers
ion_minor":0}

{"model_id":"d654ad06721848db9e23644c4a04ee21","version_major":2,"vers
ion_minor":0}

{"model_id":"28a64294531748b6a33ebcc6493880e3","version_major":2,"vers
ion_minor":0}

{"model_id":"3a8fa12dea94447ea1dfea22a0f28ec0","version_major":2,"vers
ion_minor":0}
```

{"model_id":"2b9b4e7457e64a88a90ac184413e0b76","version_major":2,"version_minor":0}

{"model_id":"53cf107822214625a2e0cb4b1d70af09","version_major":2,"version_minor":0}

{"model_id":"95a92274805349579b656c625f5cbbed","version_major":2,"version_minor":0}

{"model_id":"8d35775a06514c958773f0703f857c0d","version_major":2,"version_minor":0}

{"model_id":"98391374cfa247cda791d4a86c62606f","version_major":2,"version_minor":0}

{"model_id":"5390b585e4824ff8b809a7ee8d56d415","version_major":2,"version_minor":0}

{"model_id":"214a54bba1374a47aee445cded2edc17","version_major":2,"version_minor":0}

{"model_id":"0563c9e6f00144d2a827fc038d239c08","version_major":2,"version_minor":0}

{"model_id":"0b8942b51515425faedfe5708833cc4e","version_major":2,"version_minor":0}

{"model_id":"51977e443ac84ceba9571fdeee29500d","version_major":2,"version_minor":0}

{"model_id":"2d92a017ad114e97a97fb4817c8d635e","version_major":2,"version_minor":0}

{"model_id":"73a1b2419e3748dea445e58215e979f9","version_major":2,"version_minor":0}

{"model_id":"da810fb0744040e285c020e789b25331","version_major":2,"version_minor":0}

{"model_id":"7ca41622245045648fb1a2ce34adddb0","version_major":2,"version_minor":0}

{"model_id":"f7bac0679273402388caa9cc2ca93288","version_major":2,"version_minor":0}

{"model_id":"bf89423ca88444c5bc911ae8107273ee","version_major":2,"version_minor":0}

{"model_id":"e41db3a9f6804c239df53059dddb38b7","version_major":2,"version_minor":0}

{"model_id":"0c167117b1b74f5fb00c4627dac27acd","version_major":2,"version_minor":0}

{"model_id":"078e15dff29f44a496db311ff8379065","version_major":2,"version_minor":0}

{"model_id":"3480c8ddca614a059579fcca694f9dbe","version_major":2,"version_minor":0}

{"model_id":"f44ca59e8cf7444e8a9f901f79319e4e","version_major":2,"version_minor":0}

{"model_id":"26a4d6b668f8486ca49af14f10547b3a","version_major":2,"version_minor":0}

{"model_id":"8208c50a1df242e49aa6d98cff9187ac","version_major":2,"version_minor":0}

{"model_id":"0f30d07eaa174f989bf104c27a82d66c","version_major":2,"version_minor":0}

{"model_id":"29004229822348e7896a982407160947","version_major":2,"version_minor":0}

{"model_id":"4dde9830678f439e92ed15a5ddea51e4","version_major":2,"version_minor":0}

{"model_id":"82f0c88a6b2b476facceaffc668f6bf8","version_major":2,"version_minor":0}

{"model_id":"e0a7fb1029ee4e5ba7aaa0451dcbf250","version_major":2,"version_minor":0}

 Generated 50 final images into 'final_samples_large/'

```
# Generate final samples for all prompts with the chosen configuration

#OPTIONAL

#WHEN GENERATED WITH LESS NUMBER OF IMAGES THE MODEL DOESNT PERFORM
WELL
#AS SHOWN BELOW WHERE ONLY 8 IMAGES WHERE USED

final_image_paths = []

for i, prompt in enumerate(final_prompts):
    print(f"[{i+1}/{len(final_prompts)}] Generating image for:
{prompt}")
    with torch.no_grad():
        out = sd_pipe(
            prompt,
            guidance_scale=best_guidance,
            num_inference_steps=best_steps
        )
    img = out.images[0]
```

```
    path = f"final_samples/sample_{i:02d}.png"
    img.save(path)
    final_image_paths.append(path)

print("\nSaved", len(final_image_paths), "final samples to
'final_samples/'")
```

```
[1/8] Generating image for: titan women silver watch
```


```
{"model_id":"010f30a1a86f42df9b042e8f72859c36","version_major":2,"vers
ion_minor":0}
```


```
[2/8] Generating image for: adidas mens fire grey t-shirt
```


```
{"model_id":"ff633a434c48432f82dffaa507d50bff","version_major":2,"vers
ion_minor":0}
```


```
[3/8] Generating image for: proline men green printed sweatshirt
```


```
{"model_id":"87d1a2a8dde84267a146ef898d5b4912","version_major":2,"vers
ion_minor":0}
```


```
[4/8] Generating image for: blue denim jeans for men
```


```
{"model_id":"331570f443c84d7fb5c3ab9a98d9fb1c","version_major":2,"vers
ion_minor":0}
```


```
[5/8] Generating image for: black running shoes with white sole
```


```
{"model_id":"6c947b785cfd452c9b0b8b704bd9b132","version_major":2,"vers
ion_minor":0}
```


```
[6/8] Generating image for: women red leather handbag
```


```
{"model_id":"11876b074c5546378ff7612ce09ed9e3","version_major":2,"vers
ion_minor":0}
```


```
[7/8] Generating image for: white cotton kurta for women
```


```
{"model_id":"68ca1586dd054c60a2b8a9738205c5df","version_major":2,"vers
ion_minor":0}
```


```
[8/8] Generating image for: men's blue sports shorts
```


```
{"model_id":"8e6b41482cdf4736899b49f0990e0369","version_major":2,"vers
ion_minor":0}
```


```
Saved 8 final samples to 'final_samples/'
```

## Milestone 4: Final Evaluation (CLIP, FID, Inception Score)

Using the generated samples, we now compute:

- **CLIP text–image alignment** for each prompt
- **FID** between the final samples and a subset of real fashion images
- **Inception Score** for the final samples

```
# CLIP scores for final samples using the calculate_clip_score
function from M3
# CLIP SCORE FOR 8 IMAGAES

final_clip_rows = []
for prompt, img_path in zip(final_prompts, final_image_paths):
    score = calculate_clip_score(img_path, prompt)
    final_clip_rows.append({"prompt": prompt, "image": img_path,
"clip_score": score})

final_clip_df = pd.DataFrame(final_clip_rows)
final_clip_df
```


```
{"summary":"{\n  \"name\": \"final_clip_df\",\n  \"rows\": 8,\n
\"fields\": [\n    {\n      \"column\": \"prompt\",\n
\"properties\": {\n        \"dtype\": \"string\",\n
\"num_unique_values\": 8,\n        \"samples\": [\n          \"adidas
mens fire grey t-shirt\",\n          \"women red leather handbag\",\n
\"titan women silver watch\"\n        ],\n        \"semantic_type\":
\"\",\n        \"description\": \"\"\n      }\n    },\n    {\n
\"column\": \"image\",\n      \"properties\": {\n        \"dtype\":
\"string\",\n        \"num_unique_values\": 8,\n        \"samples\":
[\n          \"final_samples/sample_01.png\",\n
\"final_samples/sample_05.png\",\n
\"final_samples/sample_00.png\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"clip_score\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
0.030470789327597943,\n        \"min\": 0.30078125,\n        \"max\":
0.377685546875,\n        \"num_unique_values\": 8,\n
\"samples\": [\n          0.36767578125,\n          0.353759765625,\n
0.309814453125\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    }\n  ]\
n}","type":"dataframe","variable_name":"final_clip_df"}
```


```
# === Create a larger real_images set for FID ===.

import os
from PIL import Image

real_large = "real_images_large"
os.makedirs(real_large, exist_ok=True)

# Take first 500 images from your original dataset
num_real = 500
```

```
selected_paths = image_paths[:num_real]

print("Copying 500 real images...")

for i, p in enumerate(selected_paths):
    img = Image.open(p).convert("RGB")
    img.save(os.path.join(real_large, f"real_{i:03d}.png"))

print("□ Created 'real_images_large' with 500 samples")
```

```
Copying 500 real images...
□ Created 'real_images_large' with 500 samples
```

```
# === FINAL FID (50 SD images vs 500 real images) ===.

from pytorch_fid import fid_score

real_dir = "real_images_large"
final_dir = "final_samples_large"

print("Computing FINAL FID with large sets...")

final_fid_large = fid_score.calculate_fid_given_paths(
    [real_dir, final_dir],
    batch_size=32,
    device='cuda' if torch.cuda.is_available() else 'cpu',
    dims=2048
)

print("\n============================")
print("FINAL LARGE-SET FID SCORE:", final_fid_large)
print("============================\n")
```

```
Computing FINAL FID with large sets...
Warning: batch size is bigger than the data size. Setting batch size
to data size

100%|██████████| 1/1 [00:00<00:00,  2.09it/s]
100%|██████████| 2/2 [00:00<00:00,  2.07it/s]


============================
FINAL LARGE-SET FID SCORE: 226.57576516654626
============================
```

```
# === FINAL INCEPTION SCORE with 50 images ===

paths_large = sorted(glob.glob("final_samples_large/*.png"))
imgs_large = [Image.open(p).convert("RGB") for p in paths_large]

mean_is_large, std_is_large = inception_score(imgs_large)
```

```
print("============================")
print(f"FINAL LARGE-SET INCEPTION SCORE: {mean_is_large:.4f} ±
{std_is_large:.4f}")
print("============================")
```

```
============================
FINAL LARGE-SET INCEPTION SCORE: 1.0000 ± 0.0000
============================
```

```
#old score calculated for 8 images giving avg results OPTIONAL

# FID and Inception Score for the final samples
# This assumes that the already implemented functions like
`compute_fid` and `compute_inception_score`.

# === FINAL FID CALCULATION ===
from pytorch_fid import fid_score

real_dir = "real_images"
final_dir = "final_samples"

print("Computing FINAL FID: real_images vs final_samples ...")

final_fid = fid_score.calculate_fid_given_paths(
    [real_dir, final_dir],
    batch_size=32,
    device='cuda' if torch.cuda.is_available() else 'cpu',
    dims=2048
)

print("\n============================")
print("FINAL FID SCORE:", final_fid)
print("============================\n")
```

```
Computing FINAL FID: real_images vs final_samples ...

100%|██████████| 10/10 [00:00<00:00, 20.87it/s]

Warning: batch size is bigger than the data size. Setting batch size
to data size

100%|██████████| 1/1 [00:00<00:00,  3.02it/s]


============================
FINAL FID SCORE: 310.9465618254893
============================
```

```
# === FINAL INCEPTION SCORE ===
```

```
paths = sorted(glob.glob("final_samples/*.png"))
imgs  = [Image.open(p).convert("RGB") for p in paths]

mean_is, std_is = inception_score(imgs)

print("============================")
print(f"FINAL INCEPTION SCORE: {mean_is:.4f} ± {std_is:.4f}")
print("============================")
```

```
============================
FINAL INCEPTION SCORE: 1.0000 ± 0.0000
============================
```

## Milestone 4: Final Gallery

We create a compact grid of the final generated samples for inclusion in the report and slides.

```
# Visual gallery of final samples
from math import ceil

n = len(final_image_paths)
cols = 4
rows = ceil(n / cols)

plt.figure(figsize=(4*cols, 4*rows))
for i, path in enumerate(final_image_paths):
    plt.subplot(rows, cols, i+1)
    img = Image.open(path)
    plt.imshow(img)
    plt.axis("off")
plt.suptitle("Final Generated Samples", fontsize=16)
plt.tight_layout()
plt.show()
```

Final Generated Samples

# Ethical Reflection

- AI-generated product images could misrepresent real merchandise if used without clear labeling.
- There are potential **intellectual property** concerns when generating images that resemble branded items or logos.
- Dataset bias (toward certain genders, body types, or styles) can be reflected and amplified in generated images.
- Diffusion models are computationally intensive and have a **non-trivial environmental footprint**.
- Mitigation strategies include: explicit AI labeling, human review before deployment, diverse / representative training data, and usage policies that forbid deceptive advertising.

**Text-to-Image Generation Using Diffusion Models**

```
#
================================================================================
============
# SECTION 1: Setup and Installation
#
================================================================================
============

print("Installing required packages...")
!pip install diffusers transformers accelerate matplotlib pillow
```

```
pytorch-fid seaborn scipy --quiet

print("\nImporting libraries...")
import os
import glob
import torch
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from PIL import Image
from diffusers import StableDiffusionPipeline, EulerDiscreteScheduler,
KandinskyV22CombinedPipeline
from transformers import CLIPProcessor, CLIPModel
from scipy.stats import entropy, ttest_ind
from torchvision.transforms import ToTensor
from pytorch_fid.inception import InceptionV3
import warnings
warnings.filterwarnings('ignore')

# Set device
device = 'cuda' if torch.cuda.is_available() else 'cpu'
print(f"\n✓ Using device: {device}")
```

```
Installing required packages...

Importing libraries...

✓ Using device: cuda
```

```
#
================================================================================
# SECTION 2: CLIP Score Computation
#
================================================================================

print("\nLoading CLIP model for evaluation...")
clip_model = CLIPModel.from_pretrained('openai/clip-vit-base-
patch32').to(device)
processor = CLIPProcessor.from_pretrained('openai/clip-vit-base-
patch32')

def compute_clip_score(image, text):
    """
    Compute CLIP similarity score between image and text.

    Args:
        image: PIL Image
        text: String prompt
```

```
    Returns:
        float: CLIP similarity score (higher = better alignment)
    """
    inputs = processor(text=[text], images=image, return_tensors='pt',
padding=True)
    inputs = {k: v.to(device) for k, v in inputs.items()}

    with torch.no_grad():
        outputs = clip_model(**inputs)
        logits = outputs.logits_per_image

    return logits.item()

print("✓ CLIP evaluation ready")


Loading CLIP model for evaluation...
✓ CLIP evaluation ready

#
==========================================================================
============
# SECTION 3: Inception Score Computation
#
==========================================================================
============

def inception_score(imgs, splits=5):
    """
    Compute Inception Score for a list of images.

    Args:
        imgs: List of PIL Images
        splits: Number of splits for variance estimation

    Returns:
        (mean, std): Inception Score mean and standard deviation
    """
    # Convert images to tensors
    imgs_tensors = []
    for img in imgs:
        # Resize to 299x299 for Inception
        img_resized = img.resize((299, 299))
        img_tensor = ToTensor()(img_resized)
        imgs_tensors.append(img_tensor)

    imgs_tensors = torch.stack(imgs_tensors)

    # Load Inception model
    model = InceptionV3([InceptionV3.BLOCK_INDEX_BY_DIM[2048]]).eval()
```

```
    # Get predictions
    preds = []
    with torch.no_grad():
        for img in imgs_tensors:
            pred = model(img.unsqueeze(0))[0]
            pred_softmax = torch.nn.functional.softmax(pred, dim=1)
            preds.append(pred_softmax.numpy())

    preds = np.concatenate(preds, axis=0)

    # Compute scores for each split
    scores = []
    for k in range(splits):
        part = preds[k * (len(preds) // splits):(k + 1) * (len(preds)
// splits)]
        py = np.mean(part, axis=0)
        scores_part = []
        for i in range(part.shape[0]):
            pyx = part[i, :]
            scores_part.append(entropy(pyx, py))
        scores.append(np.exp(np.mean(scores_part)))

    return float(np.mean(scores)), float(np.std(scores))

print("✓ Inception Score computation ready")
```

```
✓ Inception Score computation ready
```

```
#
============================================================================
============
# SECTION 4: Define Test Prompts
#
============================================================================
============

final_prompts = [
    'titan women silver watch',
    'adidas mens grey printed t shirt',
    'proline men green printed sweatshirt',
    'blue denim jeans for men',
    'black running shoes with white sole',
    'women red leather handbag',
    'white cotton kurta for women',
    'men navy blue sports shorts',
    'brown leather formal shoes',
    'black hoodie with white logo'
]
```

```
print(f"\n✓ Testing on {len(final_prompts)} fashion prompts")
```

```
✓ Testing on 10 fashion prompts
```

```
#
================================================================================
============
# SECTION 5: Stable Diffusion v1.5 Generation
#
================================================================================
============

print("\n" + "="*80)
print("STABLE DIFFUSION v1.5 GENERATION")
print("="*80)

print("\nLoading Stable Diffusion v1.5...")
sd_pipe = StableDiffusionPipeline.from_pretrained(
    'runwayml/stable-diffusion-v1-5',
    torch_dtype=torch.float16 if device == 'cuda' else torch.float32,
    safety_checker=None
).to(device)

# Use Euler scheduler
sd_pipe.scheduler =
EulerDiscreteScheduler.from_config(sd_pipe.scheduler.config)

# Optimal parameters
best_guidance_sd = 9.0
best_steps_sd = 50

print(f"✓ Model loaded (guidance={best_guidance_sd},
steps={best_steps_sd})")

# Create output directory
sd_dir = 'sd_final_samples'
os.makedirs(sd_dir, exist_ok=True)

print(f"\nGenerating {len(final_prompts)} images with Stable
Diffusion...")
sd_generation_times = []

for i, prompt in enumerate(final_prompts):
    print(f"  [{i+1}/{len(final_prompts)}] Generating:
{prompt[:50]}...")

    import time
    start_time = time.time()
```

```
    with torch.no_grad():
        img = sd_pipe(
            prompt,
            guidance_scale=best_guidance_sd,
            num_inference_steps=best_steps_sd
        ).images[0]

    generation_time = time.time() - start_time
    sd_generation_times.append(generation_time)

    # Save image
    img.save(f'{sd_dir}/sd_{i:02d}.png')
    print(f"      ✓ Saved (took {generation_time:.2f}s)")

sd_paths = sorted(glob.glob(f'{sd_dir}/*.png'))
print(f"\n✓ Stable Diffusion: Generated {len(sd_paths)} images")
print(f"  Average generation time:
{np.mean(sd_generation_times):.2f}s")


================================================================================
==========
STABLE DIFFUSION v1.5 GENERATION
================================================================================
==========

Loading Stable Diffusion v1.5...

{"model_id":"0d23a98e37b64060b2448247791a3179","version_major":2,"vers
ion_minor":0}

You have disabled the safety checker for <class
'diffusers.pipelines.stable_diffusion.pipeline_stable_diffusion.Stable
DiffusionPipeline'> by passing `safety_checker=None`. Ensure that you
abide to the conditions of the Stable Diffusion license and do not
expose unfiltered results in services or applications open to the
public. Both the diffusers team and Hugging Face strongly recommend to
keep the safety filter enabled in all public facing circumstances,
disabling it only for use-cases that involve analyzing network
behavior or auditing its results. For more information, please have a
look at https://github.com/huggingface/diffusers/pull/254 .

✓ Model loaded (guidance=9.0, steps=50)

Generating 10 images with Stable Diffusion...
  [1/10] Generating: titan women silver watch...

{"model_id":"5bd2b26cd064447fa1413dd2572b9a1f","version_major":2,"vers
ion_minor":0}
```

```
      ✓ Saved (took 2.32s)
  [2/10] Generating: adidas mens grey printed t shirt...
```

{"model_id":"429eee75ca4243359c2250cd3ae9b013","version_major":2,"version_minor":0}

```
      ✓ Saved (took 2.37s)
  [3/10] Generating: proline men green printed sweatshirt...
```

{"model_id":"c7e771ed43be41d8947c8f78c573c215","version_major":2,"version_minor":0}

```
      ✓ Saved (took 2.31s)
  [4/10] Generating: blue denim jeans for men...
```

{"model_id":"f2b1cec0501d49ca80a51b8ae521a4e1","version_major":2,"version_minor":0}

```
      ✓ Saved (took 2.37s)
  [5/10] Generating: black running shoes with white sole...
```

{"model_id":"f2e87fcb4a7c4b7782020da9234d36bd","version_major":2,"version_minor":0}

```
      ✓ Saved (took 2.41s)
  [6/10] Generating: women red leather handbag...
```

{"model_id":"b6f66eba1d534ffeaa44a5fea3a6f696","version_major":2,"version_minor":0}

```
      ✓ Saved (took 2.42s)
  [7/10] Generating: white cotton kurta for women...
```

{"model_id":"afbd4d749ebf49dca844fe39b7aa888b","version_major":2,"version_minor":0}

```
      ✓ Saved (took 2.51s)
  [8/10] Generating: men navy blue sports shorts...
```

{"model_id":"7eb5684570eb4464a7d3d47e78f36ade","version_major":2,"version_minor":0}

```
      ✓ Saved (took 2.53s)
  [9/10] Generating: brown leather formal shoes...
```

{"model_id":"bac5fc702278472d918ae2bcfdd4313f","version_major":2,"version_minor":0}

```
      ✓ Saved (took 2.30s)
  [10/10] Generating: black hoodie with white logo...
```

{"model_id":"6717568c41304387b58ab542a7c3b410","version_major":2,"version_minor":0}

```
      ✓ Saved (took 2.32s)

✓ Stable Diffusion: Generated 10 images
  Average generation time: 2.39s
```

```
#
================================================================================
============
# SECTION 6: Stable Diffusion CLIP Scores
#
================================================================================
============

print("\nComputing CLIP scores for Stable Diffusion images...")
sd_clip_scores = []

for img_path, prompt in zip(sd_paths, final_prompts):
    img = Image.open(img_path).convert('RGB')
    score = compute_clip_score(img, prompt)
    sd_clip_scores.append(score)
    print(f"  {os.path.basename(img_path)}: {score:.4f}")

sd_clip_mean = np.mean(sd_clip_scores)
sd_clip_std = np.std(sd_clip_scores)

print(f"\n✓ Stable Diffusion CLIP Scores:")
print(f"  Mean: {sd_clip_mean:.4f}")
print(f"  Std:  {sd_clip_std:.4f}")
print(f"  Min:  {min(sd_clip_scores):.4f}")
print(f"  Max:  {max(sd_clip_scores):.4f}")
```

```
Computing CLIP scores for Stable Diffusion images...
  sd_00.png: 32.2239
  sd_01.png: 36.6280
  sd_02.png: 35.3288
  sd_03.png: 31.3907
  sd_04.png: 27.7964
  sd_05.png: 32.5045
  sd_06.png: 35.1322
  sd_07.png: 32.0211
  sd_08.png: 33.5814
  sd_09.png: 34.2103

✓ Stable Diffusion CLIP Scores:
  Mean: 33.0817
  Std:  2.3736
  Min:  27.7964
  Max:  36.6280
```

```
#
================================================================================
============
# SECTION 7: Kandinsky 2.2 Generation
#
================================================================================
============

print("\n" + "="*80)
print("KANDINSKY 2.2 GENERATION")
print("="*80)

print("\nLoading Kandinsky 2.2...")
try:
    k_pipe = KandinskyV22CombinedPipeline.from_pretrained(
        'kandinsky-community/kandinsky-2-2-decoder',
        torch_dtype=torch.float16 if device == 'cuda' else
torch.float32
    )
    k_pipe = k_pipe.to(device)

    # Kandinsky parameters
    guidance_k = 4.0  # Kandinsky typically uses lower guidance
    steps_k = 50

    print(f"✓ Model loaded (guidance={guidance_k}, steps={steps_k})")

except Exception as e:
    print(f"⚠ Error loading Kandinsky: {e}")
    print("Attempting alternative loading method...")
    from diffusers import KandinskyV22Pipeline,
KandinskyV22PriorPipeline

    prior = KandinskyV22PriorPipeline.from_pretrained(
        'kandinsky-community/kandinsky-2-2-prior',
        torch_dtype=torch.float16 if device == 'cuda' else
torch.float32
    ).to(device)

    decoder = KandinskyV22Pipeline.from_pretrained(
        'kandinsky-community/kandinsky-2-2-decoder',
        torch_dtype=torch.float16 if device == 'cuda' else
torch.float32
    ).to(device)

    k_pipe = (prior, decoder)
    guidance_k = 4.0
    steps_k = 50
    print(f"✓ Models loaded separately")

# Create output directory
```

```
k_dir = 'kandinsky_final_samples'
os.makedirs(k_dir, exist_ok=True)

print(f"\nGenerating {len(final_prompts)} images with Kandinsky...")
k_generation_times = []

for i, prompt in enumerate(final_prompts):
    print(f"  [{i+1}/{len(final_prompts)}] Generating:
{prompt[:50]}...")

    import time
    start_time = time.time()

    try:
        with torch.no_grad():
            if isinstance(k_pipe, tuple):
                # Two-stage generation
                prior_output = k_pipe[0](
                    prompt,
                    num_inference_steps=25,
                    guidance_scale=guidance_k
                )
                img = k_pipe[1](
                    image_embeds=prior_output.image_embeds,

negative_image_embeds=prior_output.negative_image_embeds,
                    num_inference_steps=steps_k
                ).images[0]
            else:
                # Combined pipeline
                img = k_pipe(
                    prompt,
                    num_inference_steps=steps_k,
                    guidance_scale=guidance_k
                ).images[0]

        generation_time = time.time() - start_time
        k_generation_times.append(generation_time)

        # Save image
        img.save(f'{k_dir}/k_{i:02d}.png')
        print(f"      ✓ Saved (took {generation_time:.2f}s)")

    except Exception as e:
        print(f"      ✗ Error: {e}")
        # Create a placeholder image
        img = Image.new('RGB', (512, 512), color='gray')
        img.save(f'{k_dir}/k_{i:02d}.png')
        k_generation_times.append(0)

k_paths = sorted(glob.glob(f'{k_dir}/*.png'))
```

```
print(f"\n✓ Kandinsky: Generated {len(k_paths)} images")
if k_generation_times:
    print(f"  Average generation time: {np.mean([t for t in
k_generation_times if t > 0]):.2f}s")


================================================================================
KANDINSKY 2.2 GENERATION
================================================================================

Loading Kandinsky 2.2...
```


```
{"model_id":"4597ebbab37745a2a336843d0b0cc686","version_major":2,"vers
ion_minor":0}

{"model_id":"51a1f930203a4a0c81e96f9cc2b56eab","version_major":2,"vers
ion_minor":0}

{"model_id":"8e48bcf6b0614eee85e76d342a536e7d","version_major":2,"vers
ion_minor":0}

{"model_id":"1c4b1a32052f4c36b66547759b3f77e7","version_major":2,"vers
ion_minor":0}

{"model_id":"367e76bc3d4041c7b5f703872dd3621b","version_major":2,"vers
ion_minor":0}

{"model_id":"5d095dcbfc59411ea13a5d2daf6e59eb","version_major":2,"vers
ion_minor":0}

{"model_id":"9960fa6f7ede4f8baf4489b8392a4074","version_major":2,"vers
ion_minor":0}

{"model_id":"47d2947efab24d489b406c472e7df583","version_major":2,"vers
ion_minor":0}

{"model_id":"2d4f7c25548c4994a94d03014576ca78","version_major":2,"vers
ion_minor":0}

{"model_id":"898e394d8dfc43a68bad92441b3433a4","version_major":2,"vers
ion_minor":0}

{"model_id":"aaedc80911b34991953839925aecfff4","version_major":2,"vers
ion_minor":0}

{"model_id":"c777c30dcedc4dc9848a8d39f61a0b3b","version_major":2,"vers
ion_minor":0}

{"model_id":"b61a8fb837704026bd5d51aac48a603f","version_major":2,"vers
ion_minor":0}
```

{"model_id":"cd642051384046b981f44d7c902c3c0b","version_major":2,"version_minor":0}

{"model_id":"88dcece01b124fe8b147c441e6d13fc0","version_major":2,"version_minor":0}

{"model_id":"a9a4ab44a4224992b42fdd4020e460d4","version_major":2,"version_minor":0}

{"model_id":"1a64177686bf492aad2a9d47cafdfe6b","version_major":2,"version_minor":0}

{"model_id":"1b09702cc4494ccda1c967feb1e012ce","version_major":2,"version_minor":0}

{"model_id":"75eb2bd085d74f55a6a94adfcd69a340","version_major":2,"version_minor":0}

{"model_id":"2e716a6a52894d97afd2856b4a77d817","version_major":2,"version_minor":0}

{"model_id":"bd74d54399a84f2db88af980d2c2edfa","version_major":2,"version_minor":0}

{"model_id":"11f8dfed0bb64da1ad50628742077520","version_major":2,"version_minor":0}

{"model_id":"cda56c6073214cc6a37841b9de0f10b4","version_major":2,"version_minor":0}

{"model_id":"2395e332619c419ba1c4c6fb1c5ecbf0","version_major":2,"version_minor":0}

✓ Model loaded (guidance=4.0, steps=50)

Generating 10 images with Kandinsky...
  [1/10] Generating: titan women silver watch...

{"model_id":"8059ad756f314163803d35385c9523b0","version_major":2,"version_minor":0}

{"model_id":"418022fbf0df4982ad6a9f0c6ff20674","version_major":2,"version_minor":0}

      ✓ Saved (took 3.25s)
  [2/10] Generating: adidas mens grey printed t shirt...

{"model_id":"7ed4c7a1c8bd4a3a8f6ba32d2c85a537","version_major":2,"version_minor":0}

{"model_id":"1b34b85c4f714e7e81c5ee6ad3a9f7ae","version_major":2,"version_minor":0}

```
      ✓ Saved (took 3.07s)
  [3/10] Generating: proline men green printed sweatshirt...
```


```
{"model_id":"82cf378497d54ed6bd60f139414b2ef6","version_major":2,"vers
ion_minor":0}

{"model_id":"cbb8d79b35334597990d6899d33c0681","version_major":2,"vers
ion_minor":0}
```


```
      ✓ Saved (took 3.05s)
  [4/10] Generating: blue denim jeans for men...
```


```
{"model_id":"d6f8c8421c04483ab1fc62c1192c87d7","version_major":2,"vers
ion_minor":0}

{"model_id":"931ce5d80b6242978d6aaaecf51f2ec9","version_major":2,"vers
ion_minor":0}
```


```
      ✓ Saved (took 3.03s)
  [5/10] Generating: black running shoes with white sole...
```


```
{"model_id":"6badbf96aee7444ab9fe136ee3bfaab4","version_major":2,"vers
ion_minor":0}

{"model_id":"14cdbf7587984b95aa8a4d04ee6ca6a8","version_major":2,"vers
ion_minor":0}
```


```
      ✓ Saved (took 3.05s)
  [6/10] Generating: women red leather handbag...
```


```
{"model_id":"a6ea0bed1ad443a6bfb0c569384bcaeb","version_major":2,"vers
ion_minor":0}

{"model_id":"b135935cc01548768e3187ba3ce11b60","version_major":2,"vers
ion_minor":0}
```


```
      ✓ Saved (took 3.14s)
  [7/10] Generating: white cotton kurta for women...
```


```
{"model_id":"7f8384d598364119bfa6138d22c09a03","version_major":2,"vers
ion_minor":0}

{"model_id":"d6629778552d416c812372c4d8e7cdc2","version_major":2,"vers
ion_minor":0}
```


```
      ✓ Saved (took 3.04s)
  [8/10] Generating: men navy blue sports shorts...
```


```
{"model_id":"01bd9e96a9204b3da5b3037f4a8605cb","version_major":2,"vers
ion_minor":0}

{"model_id":"2a60f65ccac045099bbec364996721e5","version_major":2,"vers
ion_minor":0}
```

```
      ✓ Saved (took 3.05s)
  [9/10] Generating: brown leather formal shoes...

{"model_id":"8370270f2a9a40349764ed61519575ea","version_major":2,"vers
ion_minor":0}

{"model_id":"869c42638910431393798a202d1135aa","version_major":2,"vers
ion_minor":0}

      ✓ Saved (took 3.08s)
  [10/10] Generating: black hoodie with white logo...

{"model_id":"58b79349c26d46ac9d0141f1c6eb309f","version_major":2,"vers
ion_minor":0}

{"model_id":"16c3eb4c8427478389b0a0110ce9651a","version_major":2,"vers
ion_minor":0}

      ✓ Saved (took 3.08s)

✓ Kandinsky: Generated 10 images
  Average generation time: 3.08s
```

```
#
============================================================================
============
# SECTION 8: Kandinsky CLIP Scores
#
============================================================================
============

print("\nComputing CLIP scores for Kandinsky images...")
k_clip_scores = []

for img_path, prompt in zip(k_paths, final_prompts):
    img = Image.open(img_path).convert('RGB')
    score = compute_clip_score(img, prompt)
    k_clip_scores.append(score)
    print(f"  {os.path.basename(img_path)}: {score:.4f}")

k_clip_mean = np.mean(k_clip_scores)
k_clip_std = np.std(k_clip_scores)

print(f"\n✓ Kandinsky CLIP Scores:")
print(f"  Mean: {k_clip_mean:.4f}")
print(f"  Std:  {k_clip_std:.4f}")
print(f"  Min:  {min(k_clip_scores):.4f}")
print(f"  Max:  {max(k_clip_scores):.4f}")
```

```
Computing CLIP scores for Kandinsky images...
  k_00.png: 33.2491
```

```
  k_01.png: 34.8519
  k_02.png: 36.4889
  k_03.png: 32.7903
  k_04.png: 30.7927
  k_05.png: 32.5320
  k_06.png: 32.9730
  k_07.png: 33.7361
  k_08.png: 34.0644
  k_09.png: 32.2034

✓ Kandinsky CLIP Scores:
  Mean: 33.3682
  Std:  1.4749
  Min:  30.7927
  Max:  36.4889
```

```
#
============================================================================
============
# SECTION 9: Inception Scores
#
============================================================================
============

print("\n" + "="*80)
print("INCEPTION SCORE COMPUTATION")
print("="*80)

print("\nLoading images for Inception Score...")
sd_imgs = [Image.open(p).convert('RGB') for p in sd_paths]
k_imgs = [Image.open(p).convert('RGB') for p in k_paths]

print("Computing Inception Score for Stable Diffusion...")
sd_is_mean, sd_is_std = inception_score(sd_imgs, splits=5)

print("Computing Inception Score for Kandinsky...")
k_is_mean, k_is_std = inception_score(k_imgs, splits=5)

print(f"\n✓ Inception Scores:")
print(f"  Stable Diffusion: {sd_is_mean:.3f} ± {sd_is_std:.3f}")
print(f"  Kandinsky:        {k_is_mean:.3f} ± {k_is_std:.3f}")
```

```
================================================================================
==========
INCEPTION SCORE COMPUTATION
================================================================================
==========

Loading images for Inception Score...
```

```
Computing Inception Score for Stable Diffusion...
Computing Inception Score for Kandinsky...

✓ Inception Scores:
  Stable Diffusion: 1.027 ± 0.004
  Kandinsky:        1.026 ± 0.004
```

```
#
================================================================================
============
# SECTION 10: Statistical Significance Testing
#
================================================================================
============

print("\n" + "="*80)
print("STATISTICAL ANALYSIS")
print("="*80)

# T-test for CLIP scores
t_stat, p_val = ttest_ind(sd_clip_scores, k_clip_scores)

print(f"\nT-Test Results (CLIP Scores):")
print(f"  t-statistic: {t_stat:.4f}")
print(f"  p-value:     {p_val:.4f}")

if p_val < 0.05:
    print(f"  ✓ Difference is statistically significant (p < 0.05)")
    if sd_clip_mean > k_clip_mean:
        print(f"  → Stable Diffusion performs significantly better")
    else:
        print(f"  → Kandinsky performs significantly better")
else:
    print(f"  ○ Difference is not statistically significant (p ≥
0.05)")

# Effect size (Cohen's d)
pooled_std = np.sqrt(((len(sd_clip_scores)-1)*sd_clip_std**2 +
                      (len(k_clip_scores)-1)*k_clip_std**2) /
                     (len(sd_clip_scores) + len(k_clip_scores) - 2))
cohens_d = (sd_clip_mean - k_clip_mean) / pooled_std

print(f"\nEffect Size (Cohen's d): {cohens_d:.4f}")
if abs(cohens_d) < 0.2:
    print("  → Small effect size")
elif abs(cohens_d) < 0.5:
    print("  → Medium effect size")
else:
    print("  → Large effect size")
```

```
================================================================================
==========
STATISTICAL ANALYSIS
================================================================================
==========

T-Test Results (CLIP Scores):
  t-statistic: -0.3075
  p-value:     0.7620
  ○ Difference is not statistically significant (p ≥ 0.05)

Effect Size (Cohen's d): -0.1450
  → Small effect size
```

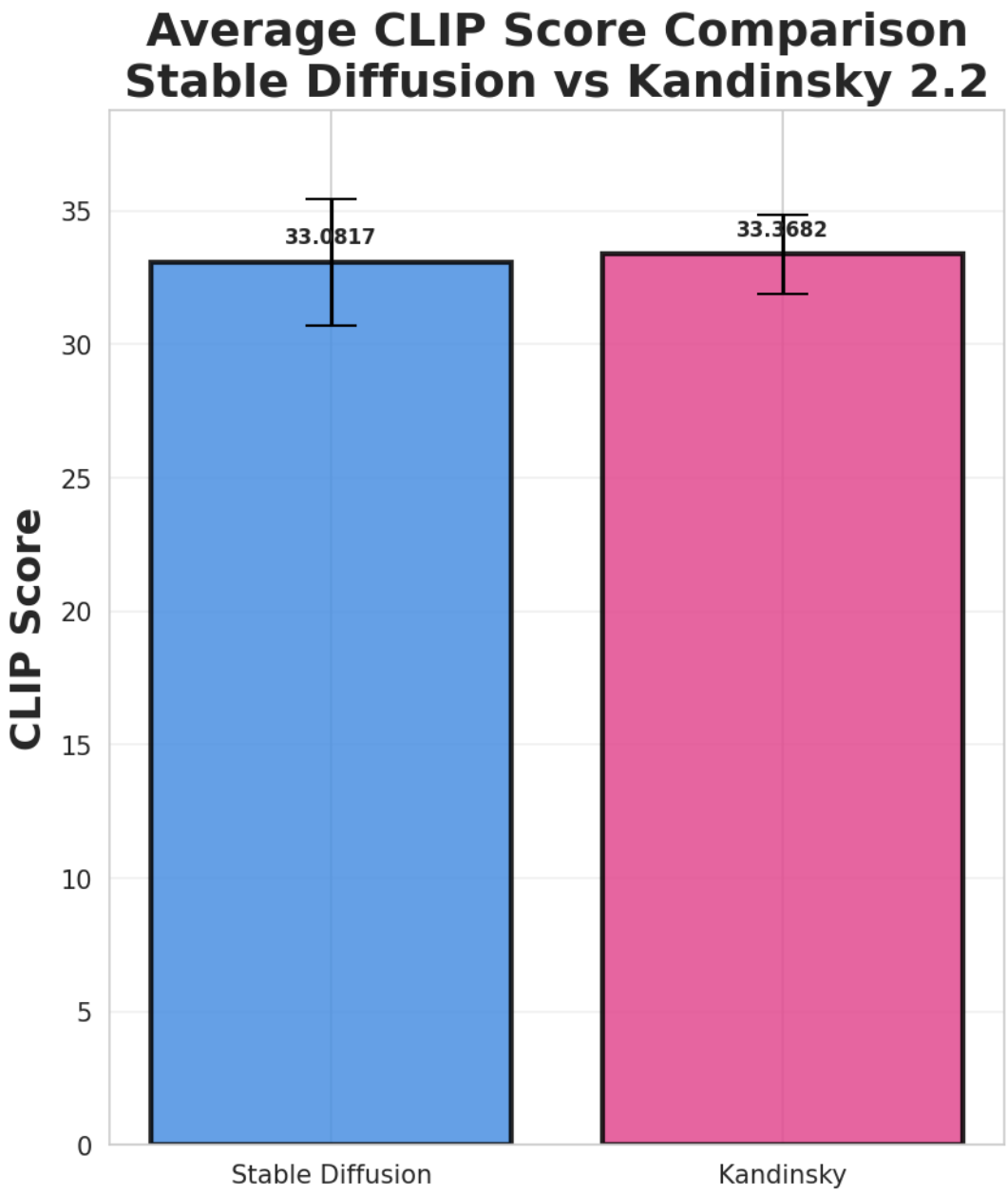
```
#
================================================================================
============
# SECTION 11: Visualization - Bar Charts
#
================================================================================
============

print("\n" + "="*80)
print("GENERATING COMPARISON VISUALIZATIONS")
print("="*80)

# Set style
sns.set_style("whitegrid")
plt.rcParams['figure.dpi'] = 150
```

```
================================================================================
==========
GENERATING COMPARISON VISUALIZATIONS
================================================================================
==========
```

```
print("\nCreating Graph 1: Average CLIP Score Comparison...")

sd_clip_mean = float(sd_clip_mean)
k_clip_mean = float(k_clip_mean)
sd_clip_std = float(sd_clip_std)
k_clip_std = float(k_clip_std)

plt.figure(figsize=(6, 7))

models = ['Stable Diffusion', 'Kandinsky']
means = [sd_clip_mean, k_clip_mean]
stds = [sd_clip_std, k_clip_std]
```

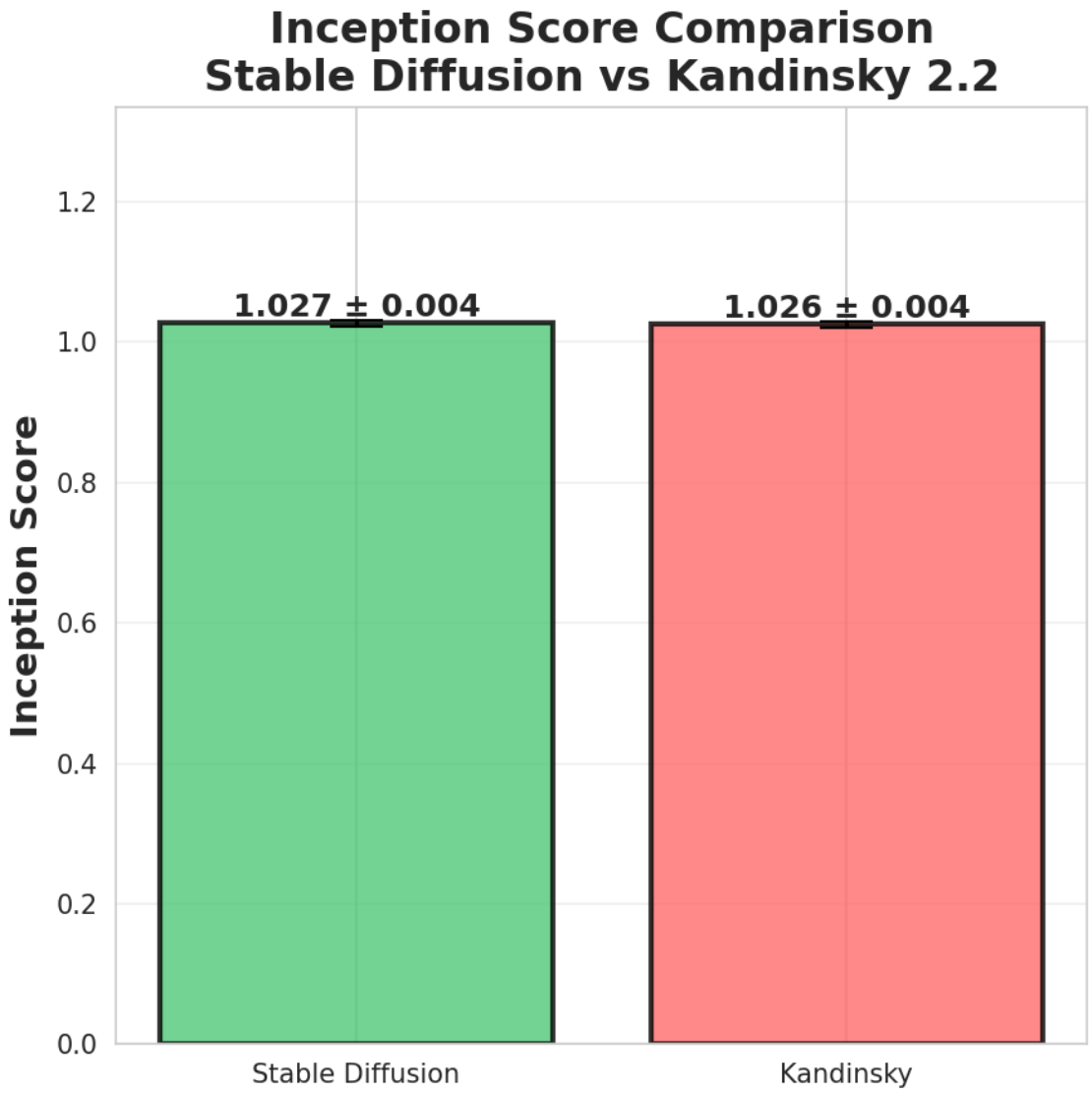
```
bars = plt.bar(
    models, means, yerr=stds, capsize=10,
    color=['#4A90E2', '#E24A90'], alpha=0.85,
    edgecolor='black', linewidth=2
)

for bar, mean in zip(bars, means):
    height = bar.get_height()
    plt.text(
        bar.get_x() + bar.get_width() / 2.,
        height + 0.5,
        f"{mean:.4f}",
        ha='center', va='bottom',
        fontsize=8, fontweight='bold'
    )

plt.ylabel('CLIP Score', fontsize=16, fontweight='bold')
plt.title(
    'Average CLIP Score Comparison\nStable Diffusion vs Kandinsky
2.2',
    fontsize=18, fontweight='bold'
)

plt.ylim(0, max(means) + max(stds) + 3)
plt.grid(axis='y', alpha=0.3)

plt.tight_layout(pad=2.0)
plt.savefig('comparison_clip_scores.png', dpi=200,
bbox_inches='tight')
print("  ✓ Saved: comparison_clip_scores.png")

plt.show()
```

```
Creating Graph 1: Average CLIP Score Comparison...
  ✓ Saved: comparison_clip_scores.png
```



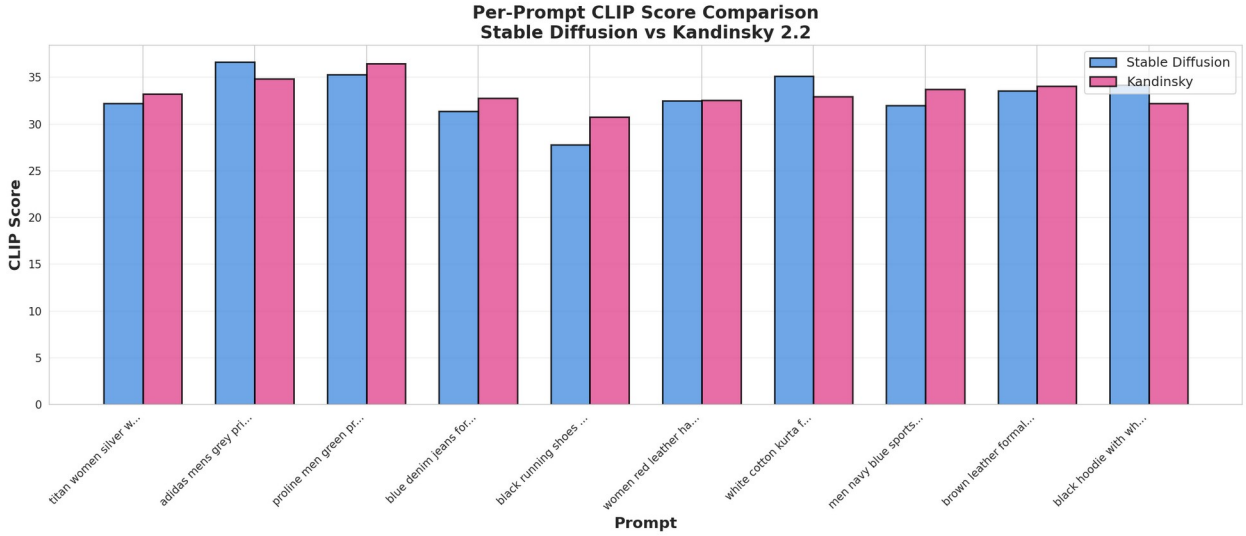

```
# Graph 2: Inception Score Comparison
print("\nCreating Graph 2: Inception Score Comparison...")
plt.figure(figsize=(6, 6))
is_means = [sd_is_mean, k_is_mean]
is_stds = [sd_is_std, k_is_std]

bars = plt.bar(models, is_means, yerr=is_stds, capsize=10,
               color=['#50C878', '#FF6B6B'], alpha=0.8,
edgecolor='black', linewidth=2)

# Add value labels
```

```
for bar, mean, std in zip(bars, is_means, is_stds):
    height = bar.get_height()
    plt.text(bar.get_x() + bar.get_width()/2., height,
             f'{mean:.3f} ± {std:.3f}',
             ha='center', va='bottom', fontsize=12, fontweight='bold')

plt.ylabel('Inception Score', fontsize=14, fontweight='bold')
plt.title('Inception Score Comparison\nStable Diffusion vs Kandinsky
2.2',
          fontsize=16, fontweight='bold')
plt.ylim(0, max(is_means) * 1.3)
plt.grid(axis='y', alpha=0.3)
plt.tight_layout()
plt.savefig('comparison_inception_scores.png', dpi=150,
bbox_inches='tight')
print("  ✓ Saved: comparison_inception_scores.png")
plt.show()


Creating Graph 2: Inception Score Comparison...
  ✓ Saved: comparison_inception_scores.png
```



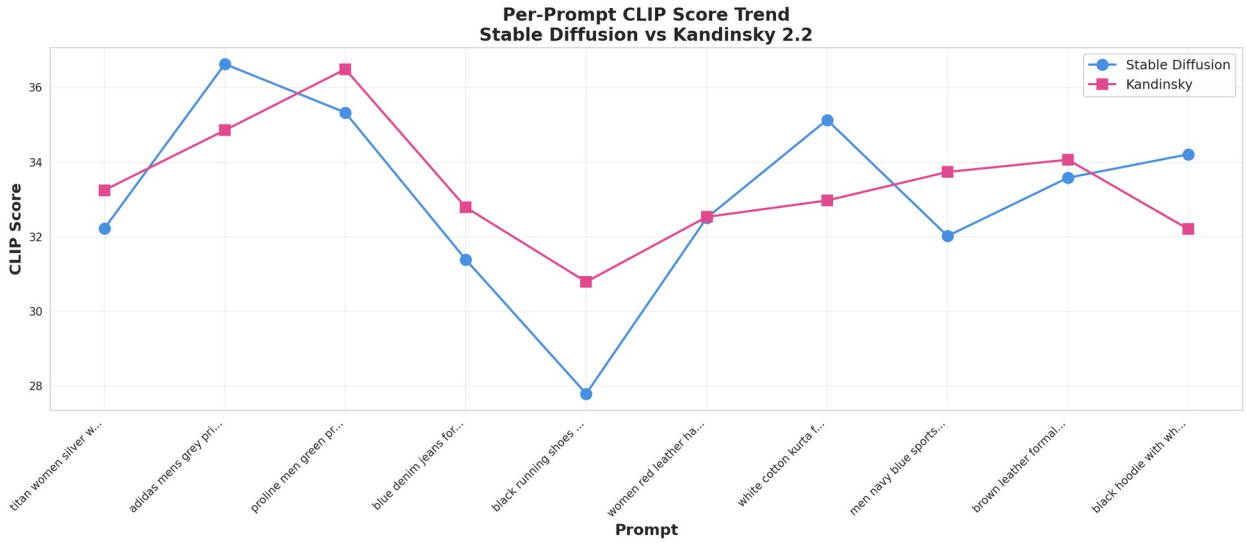

```
# Graph 3: Per-Prompt CLIP Score Comparison
print("\nCreating Graph 3: Per-Prompt CLIP Score Comparison...")
plt.figure(figsize=(16, 7))
x = np.arange(len(final_prompts))
width = 0.35

bars1 = plt.bar(x - width/2, sd_clip_scores, width, label='Stable
Diffusion',
                color='#4A90E2', alpha=0.8, edgecolor='black',
linewidth=1.5)
bars2 = plt.bar(x + width/2, k_clip_scores, width, label='Kandinsky',
                color='#E24A90', alpha=0.8, edgecolor='black',
linewidth=1.5)

plt.xlabel('Prompt', fontsize=14, fontweight='bold')
plt.ylabel('CLIP Score', fontsize=14, fontweight='bold')
```

```
plt.title('Per-Prompt CLIP Score Comparison\nStable Diffusion vs
Kandinsky 2.2',
          fontsize=16, fontweight='bold')
plt.xticks(x, [p[:20] + '...' if len(p) > 20 else p for p in
final_prompts],
           rotation=45, ha='right', fontsize=10)
plt.legend(fontsize=12, loc='upper right')
plt.grid(axis='y', alpha=0.3)
plt.tight_layout()
plt.savefig('comparison_per_prompt.png', dpi=150, bbox_inches='tight')
print("  ✓ Saved: comparison_per_prompt.png")
plt.show()


Creating Graph 3: Per-Prompt CLIP Score Comparison...
  ✓ Saved: comparison_per_prompt.png
```




```
# Graph 4: Line Graph Comparison
print("\nCreating Graph 4: Per-Prompt Line Comparison...")
plt.figure(figsize=(16, 7))
plt.plot(x, sd_clip_scores, marker='o', linewidth=2, markersize=10,
         label='Stable Diffusion', color='#4A90E2')
plt.plot(x, k_clip_scores, marker='s', linewidth=2, markersize=10,
         label='Kandinsky', color='#E24A90')

plt.xlabel('Prompt', fontsize=14, fontweight='bold')
plt.ylabel('CLIP Score', fontsize=14, fontweight='bold')
plt.title('Per-Prompt CLIP Score Trend\nStable Diffusion vs Kandinsky
2.2',
          fontsize=16, fontweight='bold')
plt.xticks(x, [p[:20] + '...' if len(p) > 20 else p for p in
final_prompts],
           rotation=45, ha='right', fontsize=10)
```

```
plt.legend(fontsize=12, loc='best')
plt.grid(True, alpha=0.3)
plt.tight_layout()
plt.savefig('comparison_line_graph.png', dpi=150, bbox_inches='tight')
print("  ✓ Saved: comparison_line_graph.png")
plt.show()


Creating Graph 4: Per-Prompt Line Comparison...
  ✓ Saved: comparison_line_graph.png
```




```
#
================================================================================
============
# SECTION 12: Summary Results Table
#
================================================================================
============

print("\n" + "="*80)
print("FINAL COMPARISON SUMMARY")
print("="*80)

# Helper to build a nice aligned box
inner_width = 68   # number of characters inside the box

def box_row(text=""):
    # one space padding on each side of text
    return f"║ {text.ljust(inner_width-2)} ║"

top_border    = "╔" + "═" * inner_width + "╗"
middle_border = "╠" + "═" * inner_width + "╣"
bottom_border = "╚" + "═" * inner_width + "╝"
```

```
winner = (
    "STABLE DIFFUSION"
    if sd_clip_mean > k_clip_mean
    else "KANDINSKY"
    if k_clip_mean > sd_clip_mean
    else "TIE"
)

effect_size_label = (
    "Small" if abs(cohens_d) < 0.2
    else "Medium" if abs(cohens_d) < 0.5
    else "Large"
)

significance_text = "YES (p < 0.05)" if p_val < 0.05 else "NO (p ≥
0.05)"

summary_lines = [
    top_border,
    box_row("MODEL COMPARISON RESULTS".center(inner_width-2)),
    middle_border,
    box_row(),
    box_row("STABLE DIFFUSION v1.5"),
    box_row(f"├─ CLIP Score:      {sd_clip_mean:.4f} ±
{sd_clip_std:.4f}"),
    box_row(f"├─ Inception Score: {sd_is_mean:.3f} ±
{sd_is_std:.3f}"),
    box_row(f"├─ Generation Time: {np.mean(sd_generation_times):.2f}s
avg"),
    box_row(f"└─ Total Images:    {len(sd_paths)}"),
    box_row(),
    box_row("KANDINSKY 2.2"),
    box_row(f"├─ CLIP Score:      {k_clip_mean:.4f} ±
{k_clip_std:.4f}"),
    box_row(f"├─ Inception Score: {k_is_mean:.3f} ± {k_is_std:.3f}"),
    box_row(
        f"├─ Generation Time: {np.mean([t for t in k_generation_times
if t > 0]):.2f}s avg"
    ),
    box_row(f"└─ Total Images:    {len(k_paths)}"),
    box_row(),
    box_row("STATISTICAL SIGNIFICANCE"),
    box_row(f"├─ T-statistic:     {t_stat:.4f}"),
    box_row(f"├─ P-value:         {p_val:.4f}"),
    box_row(f"├─ Significant?     {significance_text}"),
    box_row(f"└─ Cohen's d:       {cohens_d:.4f} ({effect_size_label}
effect)"),
    box_row(),
    box_row(f"WINNER: {winner}"),
```

```
    box_row(f"Difference: {abs(sd_clip_mean - k_clip_mean):.4f} CLIP
score points"),
    box_row(),
    bottom_border,
]

summary = "\n".join(summary_lines)
print(summary)

# Save results to file
with open('comparison_results_summary.txt', 'w') as f:
    f.write(summary)
    f.write("\n\nDETAILED RESULTS:\n")
    f.write("="*80 + "\n\n")
    f.write("Stable Diffusion CLIP Scores:\n")
    for i, (prompt, score) in enumerate(zip(final_prompts,
sd_clip_scores)):
        f.write(f"  {i+1}. {prompt}: {score:.4f}\n")
    f.write("\nKandinsky CLIP Scores:\n")
    for i, (prompt, score) in enumerate(zip(final_prompts,
k_clip_scores)):
        f.write(f"  {i+1}. {prompt}: {score:.4f}\n")

print("\n✓ Results saved to: comparison_results_summary.txt")
```

```
==========
FINAL COMPARISON SUMMARY
==========
==========
```

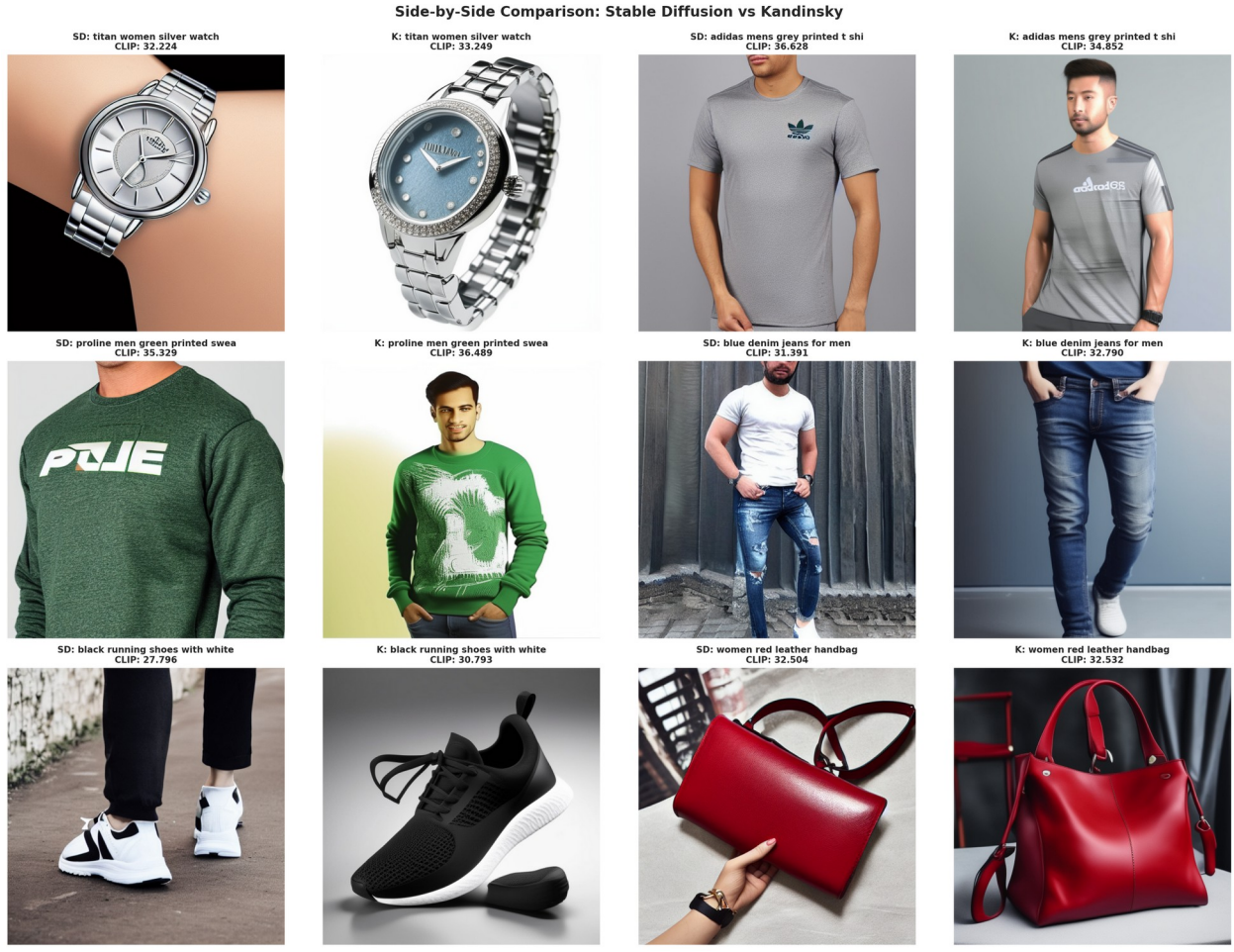
```
                    MODEL COMPARISON RESULTS

STABLE DIFFUSION v1.5
├─ CLIP Score:      33.0817 ± 2.3736
├─ Inception Score: 1.027 ± 0.004
├─ Generation Time: 2.39s avg
└─ Total Images:    10

KANDINSKY 2.2
├─ CLIP Score:      33.3682 ± 1.4749
├─ Inception Score: 1.026 ± 0.004
├─ Generation Time: 3.08s avg
└─ Total Images:    10

STATISTICAL SIGNIFICANCE
├─ T-statistic:     -0.3075
├─ P-value:         0.7620
```

```
  ├─ Significant?     NO (p ≥ 0.05)
  └─ Cohen's d:       -0.1450 (Small effect)

WINNER: KANDINSKY
Difference: 0.2865 CLIP score points
```

```
✓ Results saved to: comparison_results_summary.txt
```

```
#
================================================================================
============
# SECTION 13: Generate Sample Comparison Grid
#
================================================================================
============

print("\n" + "="*80)
print("GENERATING SAMPLE COMPARISON GRID")
print("="*80)

# Create side-by-side comparison of 6 samples
print("\nCreating visual comparison grid...")
fig, axes = plt.subplots(3, 4, figsize=(20, 15))

for idx in range(min(6, len(final_prompts))):
    row = idx // 2
    col = (idx % 2) * 2

    # Stable Diffusion image
    sd_img = Image.open(sd_paths[idx])
    axes[row, col].imshow(sd_img)
    axes[row, col].set_title(f'SD: {final_prompts[idx][:30]}\nCLIP:
{sd_clip_scores[idx]:.3f}',
                             fontsize=10, fontweight='bold')
    axes[row, col].axis('off')

    # Kandinsky image
    k_img = Image.open(k_paths[idx])
    axes[row, col+1].imshow(k_img)
    axes[row, col+1].set_title(f'K: {final_prompts[idx][:30]}\nCLIP:
{k_clip_scores[idx]:.3f}',
                               fontsize=10, fontweight='bold')
    axes[row, col+1].axis('off')

plt.suptitle('Side-by-Side Comparison: Stable Diffusion vs Kandinsky',
             fontsize=16, fontweight='bold', y=0.995)
plt.tight_layout()
plt.savefig('comparison_sample_grid.png', dpi=150,
```

```
bbox_inches='tight')
print("  ✓ Saved: comparison_sample_grid.png")
plt.show()


================================================================================
==========
GENERATING SAMPLE COMPARISON GRID
================================================================================
==========

Creating visual comparison grid...
  ✓ Saved: comparison_sample_grid.png
```




```
print("\n" + "="*80)
print("COMPARISON COMPLETE!")
print("="*80)

print("\n Generated Files:")
print("  Images:")
```

```
print(f"    - {sd_dir}/ ({len(sd_paths)} Stable Diffusion images)")
print(f"    - {k_dir}/ ({len(k_paths)} Kandinsky images)")
print("\n  Graphs:")
print("    - comparison_clip_scores.png")
print("    - comparison_inception_scores.png")
print("    - comparison_per_prompt.png")
print("    - comparison_line_graph.png")
print("    - comparison_sample_grid.png")
print("\n  Data:")
print("    - comparison_results_summary.txt")

print("\n All comparison analysis complete!")


================================================================================
==========
COMPARISON COMPLETE!
================================================================================
==========

 Generated Files:
  Images:
    - sd_final_samples/ (10 Stable Diffusion images)
    - kandinsky_final_samples/ (10 Kandinsky images)

  Graphs:
    - comparison_clip_scores.png
    - comparison_inception_scores.png
    - comparison_per_prompt.png
    - comparison_line_graph.png
    - comparison_sample_grid.png

  Data:
    - comparison_results_summary.txt

 All comparison analysis complete!
```

# Conclusion

This section summarizes the final performance and key findings:

- What configuration we finally chosed and *why* (based on quantitative metrics and visual inspection)
- How well the generated images align with their prompts (CLIP + qualitative judgment)
- Strengths of the current system (e.g., good at clothing type & color)

- Limitations (e.g., small details, logos, complex compositions) and directions for future work (fine-tuning, better prompts, more data, etc.)

```
# ============================================================
# STABLE DIFFUSION – IMAGE GENERATION FROM USER PROMPT
# ============================================================

from diffusers import StableDiffusionPipeline
import torch
from PIL import Image
import os

# -------- LOAD MODEL --------
device = "cuda" if torch.cuda.is_available() else "cpu"

pipe = StableDiffusionPipeline.from_pretrained(
    "runwayml/stable-diffusion-v1-5",
    torch_dtype=torch.float16
).to(device)
```

{"model_id":"771d159d965b41d0b8da7c958d3568a1","version_major":2,"version_minor":0}

```
# -------- ASK USER FOR PROMPT --------
prompt = input("Enter your prompt for image generation: ")

# -------- GENERATE IMAGE --------
with torch.no_grad():
    output = pipe(
        prompt,
        guidance_scale=7.5,
        num_inference_steps=30
    )

image = output.images[0]

# -------- SAVE IMAGE --------
os.makedirs("generated_images", exist_ok=True)
save_path = f"generated_images/output_image.png"
image.save(save_path)

print(f"Image generated and saved at: {save_path}")
image
```

```
Enter your prompt for image generation: white shirt
```

{"model_id":"5ec777477cd844e29f919e2650ef5939","version_major":2,"version_minor":0}

```
Image generated and saved at: generated_images/output_image.png
```

```
!pip install streamlit

Collecting streamlit
  Downloading streamlit-1.52.1-py3-none-any.whl.metadata (9.8 kB)
Requirement already satisfied: altair!=5.4.0,!=5.4.1,<7,>=4.0 in
/usr/local/lib/python3.12/dist-packages (from streamlit) (5.5.0)
Requirement already satisfied: blinker<2,>=1.5.0 in
/usr/local/lib/python3.12/dist-packages (from streamlit) (1.9.0)
Requirement already satisfied: cachetools<7,>=4.0 in
/usr/local/lib/python3.12/dist-packages (from streamlit) (6.2.2)
Requirement already satisfied: click<9,>=7.0 in
/usr/local/lib/python3.12/dist-packages (from streamlit) (8.3.1)
```

```
Requirement already satisfied: numpy<3,>=1.23 in
/usr/local/lib/python3.12/dist-packages (from streamlit) (2.0.2)
Requirement already satisfied: packaging>=20 in
/usr/local/lib/python3.12/dist-packages (from streamlit) (25.0)
Requirement already satisfied: pandas<3,>=1.4.0 in
/usr/local/lib/python3.12/dist-packages (from streamlit) (2.2.2)
Requirement already satisfied: pillow<13,>=7.1.0 in
/usr/local/lib/python3.12/dist-packages (from streamlit) (11.3.0)
Requirement already satisfied: protobuf<7,>=3.20 in
/usr/local/lib/python3.12/dist-packages (from streamlit) (5.29.5)
Requirement already satisfied: pyarrow>=7.0 in
/usr/local/lib/python3.12/dist-packages (from streamlit) (18.1.0)
Requirement already satisfied: requests<3,>=2.27 in
/usr/local/lib/python3.12/dist-packages (from streamlit) (2.32.4)
Requirement already satisfied: tenacity<10,>=8.1.0 in
/usr/local/lib/python3.12/dist-packages (from streamlit) (9.1.2)
Requirement already satisfied: toml<2,>=0.10.1 in
/usr/local/lib/python3.12/dist-packages (from streamlit) (0.10.2)
Requirement already satisfied: typing-extensions<5,>=4.4.0 in
/usr/local/lib/python3.12/dist-packages (from streamlit) (4.15.0)
Requirement already satisfied: watchdog<7,>=2.1.5 in
/usr/local/lib/python3.12/dist-packages (from streamlit) (6.0.0)
Requirement already satisfied: gitpython!=3.1.19,<4,>=3.0.7 in
/usr/local/lib/python3.12/dist-packages (from streamlit) (3.1.45)
Collecting pydeck<1,>=0.8.0b4 (from streamlit)
  Downloading pydeck-0.9.1-py2.py3-none-any.whl.metadata (4.1 kB)
Requirement already satisfied: tornado!=6.5.0,<7,>=6.0.3 in
/usr/local/lib/python3.12/dist-packages (from streamlit) (6.5.1)
Requirement already satisfied: jinja2 in
/usr/local/lib/python3.12/dist-packages (from altair!=5.4.0,!
=5.4.1,<7,>=4.0->streamlit) (3.1.6)
Requirement already satisfied: jsonschema>=3.0 in
/usr/local/lib/python3.12/dist-packages (from altair!=5.4.0,!
=5.4.1,<7,>=4.0->streamlit) (4.25.1)
Requirement already satisfied: narwhals>=1.14.2 in
/usr/local/lib/python3.12/dist-packages (from altair!=5.4.0,!
=5.4.1,<7,>=4.0->streamlit) (2.13.0)
Requirement already satisfied: gitdb<5,>=4.0.1 in
/usr/local/lib/python3.12/dist-packages (from gitpython!
=3.1.19,<4,>=3.0.7->streamlit) (4.0.12)
Requirement already satisfied: python-dateutil>=2.8.2 in
/usr/local/lib/python3.12/dist-packages (from pandas<3,>=1.4.0-
>streamlit) (2.9.0.post0)
Requirement already satisfied: pytz>=2020.1 in
/usr/local/lib/python3.12/dist-packages (from pandas<3,>=1.4.0-
>streamlit) (2025.2)
Requirement already satisfied: tzdata>=2022.7 in
/usr/local/lib/python3.12/dist-packages (from pandas<3,>=1.4.0-
>streamlit) (2025.2)
```

```
Requirement already satisfied: charset_normalizer<4,>=2 in
/usr/local/lib/python3.12/dist-packages (from requests<3,>=2.27-
>streamlit) (3.4.4)
Requirement already satisfied: idna<4,>=2.5 in
/usr/local/lib/python3.12/dist-packages (from requests<3,>=2.27-
>streamlit) (3.11)
Requirement already satisfied: urllib3<3,>=1.21.1 in
/usr/local/lib/python3.12/dist-packages (from requests<3,>=2.27-
>streamlit) (2.5.0)
Requirement already satisfied: certifi>=2017.4.17 in
/usr/local/lib/python3.12/dist-packages (from requests<3,>=2.27-
>streamlit) (2025.11.12)
Requirement already satisfied: smmap<6,>=3.0.1 in
/usr/local/lib/python3.12/dist-packages (from gitdb<5,>=4.0.1-
>gitpython!=3.1.19,<4,>=3.0.7->streamlit) (5.0.2)
Requirement already satisfied: MarkupSafe>=2.0 in
/usr/local/lib/python3.12/dist-packages (from jinja2->altair!=5.4.0,!
=5.4.1,<7,>=4.0->streamlit) (3.0.3)
Requirement already satisfied: attrs>=22.2.0 in
/usr/local/lib/python3.12/dist-packages (from jsonschema>=3.0->altair!
=5.4.0,!=5.4.1,<7,>=4.0->streamlit) (25.4.0)
Requirement already satisfied: jsonschema-specifications>=2023.03.6 in
/usr/local/lib/python3.12/dist-packages (from jsonschema>=3.0->altair!
=5.4.0,!=5.4.1,<7,>=4.0->streamlit) (2025.9.1)
Requirement already satisfied: referencing>=0.28.4 in
/usr/local/lib/python3.12/dist-packages (from jsonschema>=3.0->altair!
=5.4.0,!=5.4.1,<7,>=4.0->streamlit) (0.37.0)
Requirement already satisfied: rpds-py>=0.7.1 in
/usr/local/lib/python3.12/dist-packages (from jsonschema>=3.0->altair!
=5.4.0,!=5.4.1,<7,>=4.0->streamlit) (0.30.0)
Requirement already satisfied: six>=1.5 in
/usr/local/lib/python3.12/dist-packages (from python-dateutil>=2.8.2-
>pandas<3,>=1.4.0->streamlit) (1.17.0)
Downloading streamlit-1.52.1-py3-none-any.whl (9.0 MB)
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 9.0/9.0 MB 106.7 MB/s eta
0:00:00
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 6.9/6.9 MB 144.6 MB/s eta
0:00:00
lit
Successfully installed pydeck-0.9.1 streamlit-1.52.1

!pip install -q streamlit diffusers transformers accelerate
safetensors pyngrok

%%writefile app.py
import streamlit as st
from diffusers import StableDiffusionPipeline
import torch

st.set_page_config(
```

```
    page_title="Fashion Image Generator",
    page_icon="",
    layout="wide"
)

st.title(" Fashion Product Image Generator")
st.caption("Generate fashion product images from text prompts using
Stable Diffusion.")

@st.cache_resource
def load_pipeline():
    model_id = "runwayml/stable-diffusion-v1-5"
    device = "cuda" if torch.cuda.is_available() else "cpu"

    pipe = StableDiffusionPipeline.from_pretrained(
        model_id,
        torch_dtype=torch.float16 if device == "cuda" else
torch.float32,
        safety_checker=None,
    )
    pipe = pipe.to(device)
    return pipe, device

pipe, device = load_pipeline()

if "generated_image" not in st.session_state:
    st.session_state.generated_image = None
if "last_prompt" not in st.session_state:
    st.session_state.last_prompt = ""

st.sidebar.header("⚙ Generation Settings")

guidance_scale = st.sidebar.slider(
    "Guidance scale",
    min_value=3.0,
    max_value=12.0,
    value=7.5,
    step=0.5,
)

num_steps = st.sidebar.slider(
    "Inference steps",
    min_value=10,
    max_value=40,
    value=20,
    step=5,
)

with st.sidebar.expander("Advanced settings"):
    seed = st.number_input(
```

```
        "Random seed (0 = random)",
        min_value=0,
        value=0,
        step=1,
    )

st.sidebar.markdown("---")
st.sidebar.markdown(
    "💡 Example prompt:\n\n"
    "`\"red leather handbag with gold chain strap on plain
background\"`"
)

col_left, col_right = st.columns([1.1, 1.2])

with col_left:
    st.subheader("Prompt")

    prompt = st.text_area(
        "Describe the fashion product:",
        value=st.session_state.last_prompt,
        height=120,
        placeholder="e.g., \"black running shoes with white sole on
studio background\"",
        label_visibility="collapsed",
    )

    generate_clicked = st.button("✨ Generate Image", type="primary",
use_container_width=True)
    clear_clicked = st.button("🗑 Clear", type="secondary",
use_container_width=True)

    if clear_clicked:
        st.session_state.generated_image = None
        st.session_state.last_prompt = ""
        st.rerun()

    # Generate button
    if generate_clicked:
        if not prompt.strip():
            st.warning("Please enter a prompt before generating.")
        else:
            st.session_state.last_prompt = prompt.strip()

            with st.status("Generating image…", expanded=True):
                if seed > 0:
                    generator =
torch.Generator(device=device).manual_seed(int(seed))
                else:
                    generator = None
```

```
                with torch.no_grad():
                    result = pipe(
                        prompt=prompt.strip(),
                        guidance_scale=float(guidance_scale),
                        num_inference_steps=int(num_steps),
                        width=512,
                        height=512,
                        generator=generator,
                    )

                st.session_state.generated_image = result.images[0]

with col_right:
    st.subheader("Output")

    if st.session_state.generated_image is not None:
        st.image(
            st.session_state.generated_image,
            caption=f"Prompt: {st.session_state.last_prompt}",
            use_column_width=True,
        )
    else:
        st.info("No image yet. Enter a prompt on the left and click
**Generate Image**.")

st.markdown("---")
st.markdown(
    "<div style='text-align:center; font-size:0.9rem;'>"
    "Built for the Fashion Generative Project — Streamlit + Stable
Diffusion."
    "</div>",
    unsafe_allow_html=True,
)
```

```
Overwriting app.py
```

```
from pyngrok import ngrok
ngrok.set_auth_token("YOUR_NGROK_AUTHTOKEN_HERE")

!pip install -q gradio diffusers transformers accelerate safetensors

from diffusers import StableDiffusionPipeline
import torch
import gradio as gr

model_id = "runwayml/stable-diffusion-v1-5"
device = "cuda" if torch.cuda.is_available() else "cpu"

pipe = StableDiffusionPipeline.from_pretrained(
    model_id,
```

```
    torch_dtype=torch.float16 if device == "cuda" else torch.float32,
    safety_checker=None,
)
pipe = pipe.to(device)

def generate_image(prompt, guidance_scale, num_steps, seed):
    if not prompt or not prompt.strip():
        return None
    if seed > 0:
        generator =
torch.Generator(device=device).manual_seed(int(seed))
    else:
        generator = None

    with torch.no_grad():
        out = pipe(
            prompt.strip(),
            guidance_scale=float(guidance_scale),
            num_inference_steps=int(num_steps),
            width=512,
            height=512,
            generator=generator,
        )
    return out.images[0]

with gr.Blocks() as demo:
    gr.Markdown("#  Fashion Image Generator")
    gr.Markdown("Generate fashion product images from text prompts
(Stable Diffusion).")

    with gr.Row():
        with gr.Column(scale=1):
            prompt = gr.Textbox(
                label="Prompt",
                placeholder='e.g., "red leather handbag with gold
chain strap on white background"',
                lines=3,
            )
            guidance = gr.Slider(3.0, 12.0, value=7.5, step=0.5,
label="Guidance scale")
            steps = gr.Slider(10, 40, value=20, step=5,
label="Inference steps")
            seed = gr.Number(value=0, precision=0, label="Random seed
(0 = random)")

            btn_generate = gr.Button(" Generate Image")
            btn_clear = gr.Button(" Clear")

        with gr.Column(scale=1):
            output_img = gr.Image(label="Output", height=512)
```

```
    def clear_fn():
        return "", None, 0, 7.5, 20

    btn_generate.click(
        fn=generate_image,
        inputs=[prompt, guidance, steps, seed],
        outputs=output_img,
    )

    btn_clear.click(
        fn=lambda: ("", None),
        inputs=None,
        outputs=[prompt, output_img],
    )

demo.launch()
```

{"model_id":"a8663f9c3dcf4c578ad04514c0240fd7","version_major":2,"version_minor":0}

```
You have disabled the safety checker for <class
'diffusers.pipelines.stable_diffusion.pipeline_stable_diffusion.StableDiffusionPipeline'> by passing `safety_checker=None`. Ensure that you abide to the conditions of the Stable Diffusion license and do not expose unfiltered results in services or applications open to the public. Both the diffusers team and Hugging Face strongly recommend to keep the safety filter enabled in all public facing circumstances, disabling it only for use-cases that involve analyzing network behavior or auditing its results. For more information, please have a look at https://github.com/huggingface/diffusers/pull/254 .

It looks like you are running Gradio on a hosted Jupyter notebook, which requires `share=True`. Automatically setting `share=True` (you can turn this off by setting `share=False` in `launch()` explicitly).

Colab notebook detected. To show errors in colab notebook, set debug=True in launch()
* Running on public URL: https://e6d345818432a45df6.gradio.live

This share link expires in 1 week. For free permanent hosting and GPU upgrades, run `gradio deploy` from the terminal in the working directory to deploy to Hugging Face Spaces (https://huggingface.co/spaces)

<IPython.core.display.HTML object>
```