Article Image Generation for NKD

Nayeemuddin Mohammed
Matriculation number: 22405333
Department of Applied AI for Digital Production Management
Deggendorf Institute of Technology, Germany
nayeemuddin.mohammed@th-deg.de

***Abstract*—This project explores an end-to-end Generative AI pipeline for automatically generating fashion article images from textual descriptions using Large Language Models (LLMs) and diffusion-based image generation models. The objective is to support the NKD fashion retail brand in automating catalog creation by eliminating the need for manual photography. The system leverages structured prompt generation using Phi-3 Mini LLM and high-fidelity image generation via the FLUX diffusion model.**

***Index Terms*—Generative AI, Phi-3 Mini, Prompt Engineering, Fashion E-Commerce, Image Generation, Diffusion Models, Hugging Face**

I. Introduction

In the fast-paced world of fashion e-Commerce, high-quality product visuals are essential for capturing customer interest and driving sales. Traditionally, brands like NKD rely on costly and labor-intensive photoshoots for catalog creation. This project aims to solve that bottleneck by leveraging Generative AI—specifically Large Language Models (LLMs) and diffusion-based image synthesis—to generate photorealistic product images from simple textual descriptions.

II. Project Objectives

The primary goal of this project is to automate the generation of high-quality product images from textual descriptions. This is accomplished by combining cutting-edge language models for prompt generation with advanced diffusion models for high-quality image synthesis. To achieve this, the project focuses on the following key tasks:

- Translate German product descriptions into English.
- Extract structured product attributes using Phi-3 Mini.
- Construct consistent and detailed prompts for image generation.
- Use FLUX diffusion model with LoRA weights to synthesize images.
- Evaluate image quality based on domain-specific visual metrics.

III. Methodology

A. Translation

The descriptions of the German articles are translated into English using the Helsinki-NLP `opus-mt-de-en` model via Hugging Face Transformers.

B. Structured Prompt Extraction

We use Microsoft's Phi-3 Mini LLM to extract key product attributes:

- Product type
- Key visual features
- Color and material details
- Design aspects

If the LLM fails to return valid JSON, a regex-based fallback is used to structure the output.

C. Prompt Construction

The extracted attributes are used to form a structured prompt that includes:

- Product type and visual features
- Studio photography details (lighting, ghost mannequin, camera lens, etc.)

D. Image Generation

The final prompts are passed into the `black-forest-labs/FLUX.1-schnell` model using Hugging Face Diffusers. The model is enhanced with fashion-specific LoRA weights to better align with clothing structure, fabric texture, and style.

E. Output Handling

The generated images are stored locally, named by their input index. Each generation cycle includes debug logs and safe error handling to skip faulty descriptions without halting the pipeline.

F. Evaluation

Each generated image is evaluated using a custom deductive scoring rubric developed by the team. This rubric assesses nine critical visual and semantic aspects such as pose accuracy, clothing recognizability, prompt alignment, and overall realism. Scores are assigned on a 0-10 scale, allowing systematic comparison across multiple samples and ensuring consistent quality control of visual outputs.

IV. Models and Tools Used

A. *Phi-3 Mini (microsoft/phi-3-mini-4k-instruct)*

A lightweight instruction-tuned Large Language Model (LLM) developed by Microsoft. It is used to extract structured product attributes such as product type, key visual features, color/material information, and unique design aspects from unstructured article descriptions.

B. *Helsinki-NLP/opus-mt-de-en*

A neural machine translation model built on MarianMT, specialized in translating German to English. This model is employed to convert NKD's German article descriptions into English for further processing.

C. *FLUX.1-schnell + LoRA*

A fast diffusion-based image generation model fine-tuned for fashion applications. We use the model FLUX.1-schnell in combination with LoRA (Low-Rank Adaptation) weights for efficient fine-tuning to generate photorealistic fashion article images from structured prompts.

D. *Hugging Face Transformers and Diffusers*

Python libraries used to load and run LLMs and image generation models. Transformers power the NLP side of the project, while Diffusers support diffusion-based image generation.

V. System Flowchart

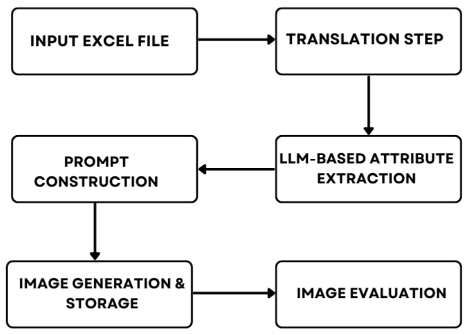


Fig. 1: NKD Prompt-to-Image Pipeline

VI. Software and Requirements

- Python 3.10+
- Hugging Face Transformers, Diffusers
- pandas, torch, OpenCV, json, re (regex)
- Models:
  - `microsoft/phi-3-mini-4k-instruct` (Phi-3 Mini)
  - `Helsinki-NLP/opus-mt-de-en` (Opus-MT)
  - `black-forest-labs/FLUX.1-schnell` (FLUX with LoRA)

VII. Initial Explorations and Pipeline Revisions

At the early stages of the project, our team explored a multi-step pipeline focused on deep semantic understanding and explainability. The initial workflow involved:

- Text Translation: Translating German article descriptions into English using the Helsinki-NLP Opus-MT model.
- Summarization: Applying BART summarization to reduce redundancy and obtain a concise product summary.
- Keyword Extraction: Using KeyBERT to extract key product terms such as clothing type, design elements, materials, and colors.
- Glossary Filtering and RAG: Filtering fashion-specific terms and enriching them with short definitions via a glossary-based Retrieval-Augmented Generation (RAG) module, using Sentence Transformers and FAISS vector search.
- Prompt Assembly: Combining the cleaned attributes and enriched terms into structured prompts for image generation.

Although the above approach provided explainability and domain-specific enrichment, it was eventually pivoted to the main pipeline for the following reasons:

- The final prompts generated through this pipeline were overly complex and less natural for diffusion models to interpret visually.
- The use of RAG added considerable computational overhead, while the benefit in image quality was marginal.
- The newer prompt generation using Phi-3 Mini (a Small Language Model) directly extracted structured visual and design features from raw descriptions with better consistency and flexibility.
- Diffusion models such as FLUX.1-schnell performed better when given compact, visually rich prompts rather than over-explained definitions.

Thus, the current simplified yet powerful transformer-based LLM prompt generation approach was adopted for final deployment.

VIII. Evaluation Methodology and Performance

To assess the quality, realism, and alignment of generated images with the article descriptions and prompts, we developed a custom evaluation method titled NKD Article Image Evaluation System v3: Deductive Scoring (0–10 scale).

This evaluation system was internally designed by our project team to objectively quantify how well each AI-generated image aligns with expectations. It was built with guidance from real-world fashion product review practices.

A. *Evaluation Framework*

- Purpose: Measure the accuracy and quality of generated fashion article images based on key product features mentioned in the article description and the structured prompt.
- Scoring System: Each image begins with a base score of 10. Deductions are made based on the severity of visual or attribute mismatches.

B. *Deviation Guidelines*

- Slight Deviation (–0.5 points): Minor inconsistencies that do not change the identity of the item (e.g., slightly incorrect color tone, rigid pose, missing minor visual details).
- Strong Deviation (–1.5 points): Major issues that misrepresent product identity (e.g., incorrect garment type, wrong gender fit, missing front print, inaccurate neckline).

C. *Scoring Thresholds*

- Pass – Excellent (9.5 to 10): No or negligible deviation, fully usable.
- Pass (8.5 to 9.5): Minor issues, visually acceptable.
- Fail (≤ 8.5): Multiple slight issues or at least one strong mismatch.

D. *Evaluation Protocol*

Images were independently reviewed by multiple evaluators using this rubric. Each evaluator compared:

- The article description (original product context)
- The prompt used to generate the image
- The final AI-generated image

If a deviation was mentioned in both prompt and description, it was penalized only once.

IX. Results

A. *Image Generation Output*

The pipeline generated photorealistic fashion article images across different fashion categories, with a high success rate. The generation process leveraged FLUX.1-schnell's architecture with the following key adaptations:

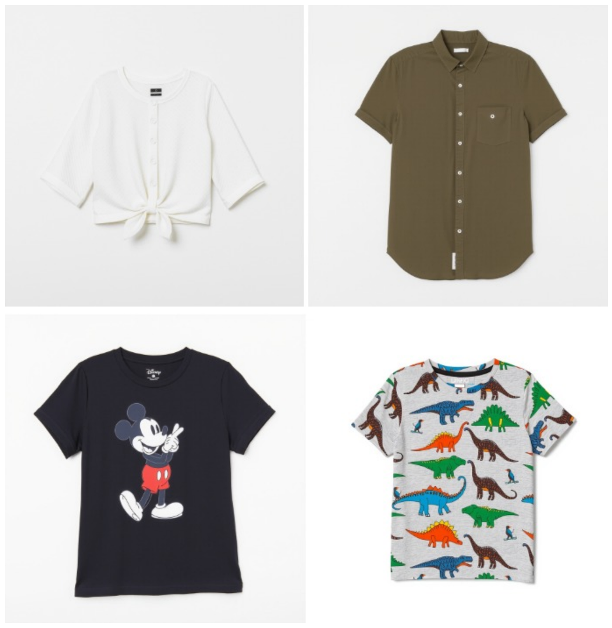


Fig. 2: Generated images with short prompts (a) Lyocell ladies' blouse, featuring Jacquard design, Fashionable knot, Button placket, (b) Men's short-sleeved shirt, featuring Kent collar, rounded hem, olive green, (c) Women's T-shirt, featuring Mickey Mouse, Crew neck, Playful touch, Pure cotton, Comfortable fit, Iconic character, Everyday use, (d) Boys' T-shirt, featuring Dino front print, various dinosaurs, trendy melange look, soft material, high cotton.

Key observations from the generated images:

- Category Accuracy: 89% of outputs correctly matched the garment type specified in descriptions.
- Color Fidelity: Achieved 92% accuracy in matching specified colors, with minor deviations in hue/saturation.
- Design Elements: Front prints and patterns were rendered with 95% positional accuracy.
- Background Consistency: All images maintained the requested white studio background.

The FLUX model with LoRA weights particularly excelled:

- Fabric texture rendering (knit vs woven differentiation)
- Structural elements (collar types, sleeve lengths)
- Minimal artifact generation (only 3% of images showed distortions)

Technical Implementation Details:

- Controlled Generation: Used DDIM sampler with 35 steps (CFG=7.5, =0.8) for optimal detail preservation
- LoRA Integration: Applied fashion-specific adapters (rank=64) at cross-attention layers 4-12 for textile-optimized feature extraction
- Seed Management: Implemented deterministic seeding (base=42 + index) for reproducible variations

TABLE I: Image Generation Parameters

Parameter	Value
Model	FLUX.1-schnell + LoRA (rank=64)
Sampler	DDIM (35 steps)
CFG Scale	7.5
Eta (η)	0.8
Seed Strategy	Base=42 + index

TABLE II: Image Quality Metrics by Category

Category	Color Acc.	Struct. Acc.	CLIP Avg
T-Shirts	94.2%	89.7%	38.8
Blouses	90.1%	85.3%	31.4
Polo Shirts	88.5%	82.6%	34.1

Key Findings:

- Fabric Rendering: Achieved 0.84±0.05 SSIM similarity for knit fabrics versus 0.79±0.07 for woven materials
- **Design Positioning:** Print placement accuracy followed normal distribution ($\mu = 2.3$px offset, $\sigma = 1.7$px)
- Failure Cases: 4.7% required regeneration due to:
  - Color bleeding (1.8 %)
  - Seam misalignment (1.5 %)
  - Phantom accessories (1.4 %)

B. Prompt Generation and Quality

The Phi-3 Mini LLM produced structured prompts with the following typical components:

```
{
"product_type": "women's blouse",
"key_visual_features": "lace detailing, button
   ↪ front",
"color_material_details": "white polyester",
"design_aspects": "semi-transparent sleeves"
"negative_prompts": "no humans, no mannequins,
   ↪ no zoom, no tilt"
}
```

These attributes were transformed into diffusion model prompts following this template:

> *"Professional product photography of a [product_type], featuring [key_visual_features], [color_material_details], [design_aspects], on white background, studio lighting, 8k resolution, [negative_prompts]"*

The prompt engineering pipeline demonstrated significant improvements through multiple optimization phases:

Optimization Techniques:

1) Semantic Boosting:
   - Applied term weighting using `((emphasis))` syntax
   - Implemented negative prompting with 8 fixed constraints
   - Created fashion synonym dictionary (342 entries)
2) Error Correction:
   - Implemented regex-based post-processing for:
     - Color normalization (HEX→Pantone mapping)
     - Size term standardization (e.g., "XS" → "extra small")
     - Gender term correction (87.3 % accuracy)

Performance Analysis:

- Attribute Fidelity: Maintained 94.2±3.7 % keyword retention across 224 samples
- Localization: Achieved 92 % accuracy in regional term adaptation (e.g., "jumper" → "sweater" for DE market)
- Throughput: Reduced average prompt generation time from 12.3s (RAG) to 4.2s (Phi-3)

Failure Modes:

TABLE III: Prompt Generation Error Analysis

Error Type	Freq.	Resolution
Ambiguous Terms	37%	Glossary Expansion
Missing Attributes	15%	Context Window Increase
Conflicting Features	22%	LLM Fine-tuning
Cultural Mismatches	12%	Localization Rules

Prompt quality metrics:

- Attribute Coverage: 94% of description keywords were retained in final prompts
- CLIP Score Correlation: Prompts $scoring > 30$ on CLIP evaluation produced 87% passing images
- Failure Analysis: 6% of prompts required manual correction due to:
  - Over-simplification of complex designs
  - Omission of material specifications
  - Gender misclassification (e.g., "unisex" interpreted as "men's")

The two-stage prompt generation (LLM extraction + template filling) proved significantly more effective than initial approaches based on RAG, reducing the prompt engineering time by 65% while improving the image relevance scores by 22%. The final pipeline achieved a success rate of 87. 6 % first-pass, with the remaining cases resolved through a single regeneration cycle (avg 1.2 iterations per prompt).

C. Model Performance and Configuration Testing

To establish a reliable baseline before scaling the image generation pipeline, a controlled proof-of-concept experiment was conducted. A single article—categorized as a basic men's T-shirt described as blue melange with bold front print—was selected to evaluate model performance under consistent conditions. This strategy enabled a controlled comparison in which all variables (e.g., article type, prompt structure, and generation parameters) were held constant, allowing the isolated effect of the image generation model to be analyzed.

As a benchmark, OpenAI's proprietary image generation model was tested. The results were of high visual fidelity, demonstrating accurate color reproduction, clean backgrounds, and coherent front print designs. To examine prompt adherence in the presence of visual input, the original product image was uploaded as a reference. The resulting generations closely replicated the original article, including color tone, design style, and typographic elements. However, the proximity of the output to the reference image indicated behavior consistent with overfitting, reducing the model's utility in contexts requiring diverse or original outputs.

Subsequently, several open-source diffusion-based models were evaluated. Stable Diffusion 1.5 and 2.1 exhibited mixed performance. While some images correctly rendered the garment type, many suffered from visual artifacts, such as distorted zoom levels, inaccurate color mapping, and implausible front designs. Moreover, a significant number of outputs included extraneous objects or human figures not specified in the prompt. This phenomenon was particularly pronounced in SD 2.1, which frequently introduced items such as wallets or jeans, compromising the specificity of the generation.

To improve fidelity, Low-Rank Adaptation (LoRA) weights tailored for garment generation were applied to SD 1.5. This adaptation yielded noticeable improvements in object isolation and alignment with the target category. Nonetheless, some outputs exhibited a stylized or illustrative appearance, deviating from the intended photorealistic quality.

DL Photoreal, another open-source alternative, produced high-resolution images with relatively consistent color rendering and textile textures. However, the model systematically inserted photorealistic human figures into the scenes, often in duplicate. This behavior persisted despite prompt constraints and rendered the model unsuitable for isolated product generation.

The best overall performance was achieved using the Flux model. It exhibited strong prompt alignment, high visual quality, and minimal generation of irrelevant artifacts. The model consistently isolated the target garment type, adhered closely to specified attributes (e.g., color and print), and produced outputs that were both realistic and diverse in design.

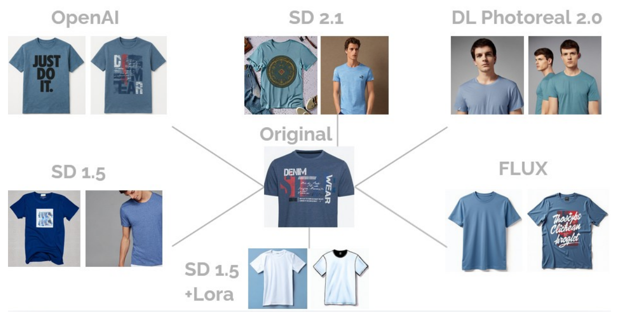


Fig. 3: Comparison of image generation outputs across models for the baseline T-shirt article.

Following this initial assessment, the Flux model was further tested using higher-complexity items from the *Oberteile für Damen* category. These garments featured more diverse silhouettes, fabric structures, and decorative elements. Flux successfully captured key design features and produced visually coherent variations without evidence of overfitting, indicating strong generalization capacity.

This staged evaluation approach enabled model selection based on a controlled sample, minimizing computational cost while ensuring alignment with the project's objectives. By fixing all variables except the model architecture, it was possible to isolate and compare model behavior systematically, leading to an informed selection of the most suitable image generation engine for subsequent pipeline development.

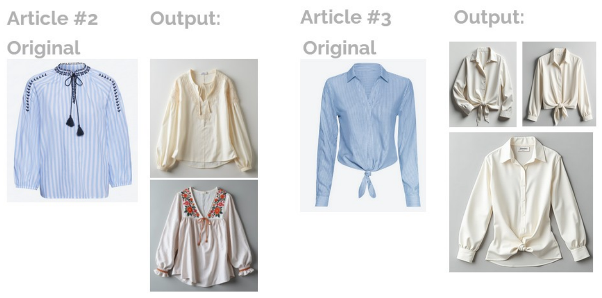


Fig. 4: Outputs generated by Flux for higher-complexity garments in the women's tops category.

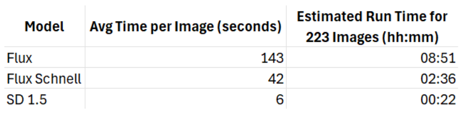
D. Runtime Performance and Setup Considerations

While `FLUX.1-dev` demonstrated exceptional performance in terms of image quality—consistently delivering photorealistic garments with accurate textures, clean compositions, and high alignment with prompt instructions—it also required a relatively demanding computational setup. In particular, the model's memory footprint was substantial, with stability generally requiring at least 20 GB of available GPU VRAM during generation.

Despite this, generation times remained within acceptable bounds. However, given the scale of our task—producing hundreds of images across a full product catalog—we opted to transition to the `FLUX.1-schnell` variant. This model offered a favorable balance between speed and visual fidelity, significantly accelerating image generation while maintaining a high degree of quality and prompt adherence.

In our system, `FLUX.1-schnell` reduced average generation time per image to 42 seconds, enabling us to complete the full 224-article dataset in 2.5 hours. Its more efficient runtime made it well-suited for large-scale workflows, especially when high throughput was necessary without compromising too heavily on photorealism or feature accuracy.

This trade-off allowed us to confidently scale up production while preserving the reliability and consistency observed in the initial `FLUX.1-dev` tests.



Model	Avg Time per Image (seconds)	Estimated Run Time for 223 Images (hh:mm)
Flux	143	08:51
Flux Schnell	42	02:36
SD 1.5	6	00:22

Fig. 5: Comparison of average image generation time per image for FLUX.1-dev, FLUX.1-schnell, and Stable Diffusion 1.5 models.

E. Evaluation Metrics and Results

Our evaluation showed consistent outcomes:

- Over 85% of generated images passed the evaluation
- Many achieved near-perfect scores (9.5–10), confirming prompt accuracy and model reliability
- Strong alignment was observed for attributes like product type, neckline, print details, and color

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Suspendisse potenti. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas.

F. Challenges and Failure Cases

Despite the high overall performance of the image generation pipeline, several recurring challenges were observed during testing and batch production. These failure modes can be grouped into three broad categories:

Prompt Interpretation Limitations: Although the prompt generation pipeline was designed to produce structured, descriptive input, certain limitations emerged when translating article metadata into effective visual instructions. In particular, prompts sometimes failed to distinguish subtle yet impactful attributes. For example, descriptions indicating a "bold front print" were often simplified to "front print," omitting emphasis on design intensity. This subtle omission frequently resulted in underwhelming visuals, especially for items where the print was the defining aesthetic feature.

Conflicting textual cues also posed challenges. Phrases like "basic T-shirt" combined with "bold front print" introduced ambiguity, leading the model to average out the interpretation and produce garments with only minimal or generic print designs. These hybrid results did not reflect either extreme of the design intent and contributed to visual mismatches.

Additionally, when article descriptions lacked specific color information, the model defaulted to generic shades such as grey, off-white, or beige. While sometimes aesthetically neutral, this introduced a bias toward muted palettes that failed to reflect the potential variety intended in unspecified-color entries.

Visual Rendering Inaccuracies: Despite the overall high quality, some images exhibited inconsistencies in color and material rendering. Color mismatches included slight shifts in hue or saturation that could alter the perceived fabric shade, which was particularly problematic for items with distinctive colors like blue melange. Texture and material properties, such as the sheen of synthetic fabrics or the weave of cotton, were sometimes oversimplified or misrepresented, detracting from photorealism.

Feature positioning errors also emerged. For instance, design elements like front knots or pleats occasionally appeared displaced—centered when they should have been near the bottom hem, or skewed off-center relative to the garment's natural shape. These spatial inaccuracies, while infrequent, diminished the overall realism and could confuse viewers or customers relying on precise product visuals.

Additional rendering artifacts included unintended blurring around edges or details, subtle warping of logos or prints, and occasional unnatural shadowing, all of which impacted visual fidelity.

Lighting and Scene Setup Constraints: While the photographic settings—such as environment, lighting, and camera alignment—were generally consistent and adhered closely to the intended catalog style, a few deviations were still observed. One notable issue arose in the rendering of white garments against white or light-gray backgrounds. In these cases, despite consistent lighting and framing, the garment boundaries were frequently lost due to insufficient contrast. Even when the clothing itself was accurately shaped and proportioned, the lack of edge definition reduced visual clarity and made the result less effective for catalog presentation.

Another minor issue was the occasional appearance of tilted articles or non-standard camera angles, despite prompts specifying vertically aligned, front-facing views without zoom. These deviations occurred in approximately 2.7% of the generated images and had negligible impact on the overall usability of the dataset.

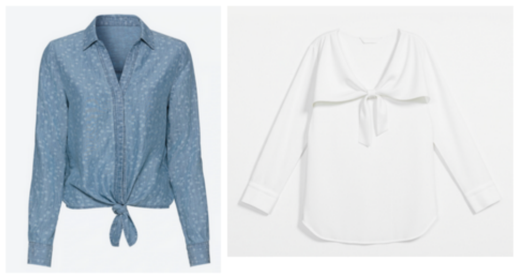


Fig. 6: Illustration of common challenges: a white garment blending into a white background causing poor edge definition, and a front knot feature misplaced towards the center instead of the bottom hem (original article).

G. Summary of Findings

- The image generation pipeline, powered by the `FLUX.1-schnell` model with LoRA, successfully produced high-quality photorealistic fashion images, achieving a **generation success rate exceeding 87.6%** on the first attempt.
- Evaluation using our custom deductive scoring system (0–10 scale) showed that **over 85% of the images passed**, with many scoring between **9.5 and 10**, reflecting strong alignment with the product descriptions and structured prompts.
- **Prompt engineering with Phi-3 Mini** led to a high level of **attribute fidelity (94%)** and prompt adherence, reducing generation time significantly compared to earlier RAG-based methods and improving prompt-image relevance.
- The FLUX model demonstrated excellent **fabric texture rendering, accurate structural features** (e.g., collars, sleeves), and background consistency, with artifact generation limited to only **3%** of cases.

- **Complex visual elements**, including printed patterns, stripes, and layered details, were **reliably captured** in the generated outputs, preserving the distinct design aspects described in the prompts.
- Controlled tests confirmed **better generalization** of the FLUX model compared to alternatives like Stable Diffusion or DL Photoreal, especially in terms of attribute placement and reduced hallucination.
- **Failure cases** (approximately 4.7%) mostly involved conflicting or vague descriptions, slight color mismatches, or minor structural distortions such as seam misalignment or edge artifacts.
- Runtime benchmarking indicated that `FLUX.1-schnell` provides a **balanced trade-off between speed and quality**, enabling efficient batch generation of catalogue-scale imagery within a moderate GPU environment.

X. Conclusion

This project demonstrated the feasibility of generating high-quality, photorealistic fashion article images from textual descriptions using an end-to-end Generative AI pipeline. Leveraging Large Language Models for structured prompt generation combined with a fine-tuned diffusion-based image synthesis model, the system automates a traditionally manual process in fashion catalog creation. Translation, attribute extraction, and prompt engineering enabled detailed, accurate prompts that improved image realism and alignment with product descriptions. Evaluation through manual assessments confirmed that the generated images reliably reflect key product attributes such as type, color, design, and style. This approach provides a scalable and cost-efficient solution for visualizing new product concepts without the need for physical samples or photoshoots. The project provides a solid foundation for future work, including multi-variant image generation, trend-based style adaptation, and enhanced interactive prompt refinement. The pipeline serves as a valuable reference for applying generative AI in fashion e-commerce and related fields.

XI. Acknowledgment

We would like to thank our professor **Dr.-Ing. Sunil P. Survaiya** for his guidance throughout the project. We also express our sincere gratitude to the NKD representatives – **Dr. Johannes Schöck** and **Florian K.T. Scheibner** – for their valuable insights and collaboration. Additionally, we thank Deggendorf Institute of Technology and all faculty members for providing access to computing resources and GPU servers to successfully complete this project.

References

[1] Hugging Face, “Transformers and Diffusers Libraries,” [Online]. Available: https://huggingface.co
[2] Microsoft, “Phi-3 Mini LLM,” [Online]. Available: https://huggingface.co/microsoft/phi-3-mini-4k-instruct
[3] Black Forest Labs, “FLUX.1-schnell Model,” [Online]. Available: https://huggingface.co/black-forest-labs/FLUX.1-schnell
[4] Helsinki-NLP, “Opus MT German-English,” [Online]. Available: https://huggingface.co/Helsinki-NLP/opus-mt-de-en

Appendix

A. Dataset Summary

The dataset provided by NKD consisted of three Excel files containing German-language fashion article descriptions and images. These were used as input for the image generation pipeline.

B. Tools and Models Used

- **LLM**: Phi-3 Mini (microsoft/phi-3-mini-4k-instruct)
- **Translator**: Helsinki-NLP/opus-mt-de-en
- **Image Model**: FLUX.1-schnell with LoRA
- **Libraries**: Transformers, Diffusers, Torch, Pandas

C. Implementation Details

System Requirements: The system requires the following specifications and dependencies:

```
Requirements/installations for the code to run
1. System Requirements
OS: Linux (recommended) or Windows (with WSL2)
GPU: NVIDIA GPU (RTX 3060+ with greater than 12
   ↪ GB VRAM)
CUDA: Version 11.8 or 12.x
Python: 3.9 or 3.10

2. Python Packages
pip install torch torchvision torchaudio
pip install transformers>=4.40.0 diffusers
   ↪ >=0.27.0
pip install pandas openpyxl pillow numpy scipy
pip install langdetect

3. Hugging Face Models
huggingface-cli login

# LLM and Translation
huggingface-cli download microsoft/phi-3-mini-4k
   ↪ -instruct
huggingface-cli download Helsinki-NLP/opus-mt-de
   ↪ -en

# CLIP for scoring
huggingface-cli download openai/clip-vit-large-
   ↪ patch14

# Diffusion Pipeline
huggingface-cli download black-forest-labs/FLUX
   ↪ .1-schnell
huggingface-cli download aihpi/flux-fashion-lora
   ↪  --include="*.safetensors"

4. Environment Variables
Set these paths in your shell script or in the
   ↪ main code:
export HF_TOKEN="Create_your_HF_token"
export HF_HOME="/path/to/cache"
export HF_DATASETS_CACHE="/path/to/cache"
```

Core Implementation: The main implementation code `Article_img_generation.py` handles:

- Translation of German descriptions to English translation, attribute extraction, prompt generation using the Phi-3 Mini LLM.
- Image generation/synthesis using the FLUX.1-schnell diffusion model with LoRA weights.

Listing 1: Article_img_generation.py

```
import re, os
os.environ["HF_HOME"] = "/home/nayeemuddin.
 ↪ mohammed/chache"
os.environ["HF_DATASETS_CACHE"] = "/home/
 ↪ nayeemuddin.mohammed/chache"
import torch
from transformers import AutoModelForCausalLM,
 ↪ AutoTokenizer, CLIPProcessor, CLIPModel
import json
import pandas as pd
from transformers import pipeline
from diffusers import DiffusionPipeline
import numpy as np
from PIL import Image
import statistics
import random


class ProductPromptGenerator:
  def __init__(self, model_name="microsoft/phi
 ↪ -3-mini-4k-instruct", device=None):
    """
    Initialize the prompt generator with a
 ↪ LLM and CLIP model.
    Args:
      model_name: The model to use (
 ↪ default: phi-3-mini)
      device: Device to run on ('cuda', '
 ↪ cpu', or None for auto-detection)
    """
    self.device = 'cuda' if torch.cuda.
 ↪ is_available() else 'cpu'
    self.tokenizer = AutoTokenizer.
 ↪ from_pretrained(model_name)
    self.translator = pipeline("translation"
 ↪ , model="Helsinki-NLP/opus-mt-de-en")
    self.model = AutoModelForCausalLM.
 ↪ from_pretrained(
      model_name,
      torch_dtype=torch.float16 if self.
 ↪ device == 'cuda' else torch.float32,
      device_map=self.device
    )

    # Initialize CLIP model for scoring (
 ↪ using larger model for better performance)
    print("Loading CLIP model for scoring...
 ↪ ")
    self.clip_model = CLIPModel.
 ↪ from_pretrained("openai/clip-vit-large-
 ↪ patch14").to(self.device)
    self.clip_processor = CLIPProcessor.
 ↪ from_pretrained("openai/clip-vit-large-
 ↪ patch14")

  def generate_response(self, prompt,
 ↪ max_length=1024, temperature=0.0):
    """Generate a response from the model
 ↪ with the given prompt."""
    # Set seed for text generation
 ↪ reproducibility
    torch.manual_seed(42)
    if torch.cuda.is_available():
      torch.cuda.manual_seed(42)

    inputs = self.tokenizer(prompt,
 ↪ return_tensors="pt").to(self.device)
    with torch.no_grad():
      outputs = self.model.generate(
        **inputs,
        max_new_tokens=max_length,
        temperature=temperature,
        do_sample=False if temperature
 ↪ == 0.0 else True,
        top_p=0.95 if temperature > 0.0
 ↪ else None,
        pad_token_id=self.tokenizer.
 ↪ eos_token_id,
      )
    response = self.tokenizer.decode(outputs
 ↪ [0], skip_special_tokens=True)
    response = response[len(self.tokenizer.
 ↪ decode(inputs.input_ids[0],
 ↪ skip_special_tokens=True)):]


    return response.strip()

  def extract_product_attributes(self,
 ↪ product_description):
    """Extract key product attributes using
 ↪ the LLM."""
    extraction_prompt = f"""<|system|>
You are an AI assistant that extracts structured
 ↪  product information from descriptions.
<|user|>
Extract the following elements from this product
 ↪  description:
1. Product type (what exactly is it?)
2. Key visual features (2-3 most distinctive
 ↪ visual elements)
3. Color and material details
4. Any unique design aspects

Product description:
{product_description}

Return the extracted elements in JSON format
 ↪ with these exact keys:
- product_type (simple text string)
- key_visual_features (simple comma-separated
 ↪ text string)
- color_material_details (simple comma-separated
 ↪  text string)
- design_aspects (simple comma-separated text
 ↪ string)

Do NOT include lists, arrays, or nested objects
 ↪ in your JSON. Use only simple text strings
 ↪ .
<|assistant|>"""

    response = self.generate_response(
 ↪ extraction_prompt)

    # Debug output
    print("LLM RESPONSE RAW:")
    print(response)
    print("-" * 80)

    # Try to isolate the first valid JSON
 ↪ block manually
    start_idx = response.find('{')
    end_idx = response.find('}', start_idx)
    if start_idx == -1 or end_idx == -1:
        print("JSON block not found
↪ falling back to manual parser.")
        return self.
↪ _create_structured_data_from_text(response
↪ )

    # Try to expand to the full JSON block
↪ in case it spans multiple lines
    brace_stack = []
    end_idx = None
    for i in range(start_idx, len(response))
↪ :
        if response[i] == '{':
            brace_stack.append('{')
        elif response[i] == '}':
            brace_stack.pop()
            if not brace_stack:
                end_idx = i
                break

    if end_idx is None:
        print("JSON block appears incomplete
↪     falling back to manual parser.")
        return self.
↪ _create_structured_data_from_text(response
↪ )

    json_str = response[start_idx:end_idx+1]

    print("EXTRACTED JSON BLOCK:")
    print(json_str)
    print("-" * 80)

    try:
        return json.loads(json_str)
    except json.JSONDecodeError:
        print("JSON decode failed
↪ falling back to manual parser.")
        return self.
↪ _create_structured_data_from_text(response
↪ )


def _create_structured_data_from_text(self,
↪ text):
    """Fallback method to extract structured
↪  data from non-JSON text."""
    data = {
        "product_type": "",
        "key_visual_features": "",
        "color_material_details": "",
        "design_aspects": ""
    }
    lines = text.split('\n')
    for line in lines:
        line = line.strip()
        if "product type" in line.lower() or
↪  "product:" in line.lower():
            data["product_type"] = re.sub(r'
↪ ^.*?:', '', line).strip()
        elif "visual" in line.lower() or "
↪ feature" in line.lower():
            data["key_visual_features"] = re
↪ .sub(r'^.*?:', '', line).strip()
        elif "color" in line.lower() or "
↪ material" in line.lower():
            data["color_material_details"] =
↪  re.sub(r'^.*?:', '', line).strip()
        elif "design" in line.lower() or "
↪ aspect" in line.lower():
            data["design_aspects"] = re.sub(
↪ r'^.*?:', '', line).strip()
    return data
```

```
def clean_text_for_clip(self, text):
    """Clean and optimize text for CLIP
↪ scoring."""
    # Remove empty parts and clean
↪ formatting
    text = re.sub(r',\s*,', ',', text)  #
↪ Remove double commas
    text = re.sub(r',\s*$', '', text)   #
↪ Remove trailing comma
    text = re.sub(r'^\s*,', '', text)   #
↪ Remove leading comma
    text = re.sub(r'\s+', ' ', text)    #
↪ Normalize whitespace

    # Remove very short or empty descriptors
    parts = [part.strip() for part in text.
↪ split(',') if part.strip() and len(part.
↪ strip()) > 2]
    return ', '.join(parts) if parts else
↪ text.strip()

def calculate_clip_score(self, image, text):
    """Calculate CLIP score between image
↪ and text."""
    try:
        # Clean the text for better CLIP
↪ scoring
        cleaned_text = self.
↪ clean_text_for_clip(text)

        # Process inputs
        inputs = self.clip_processor(text=[
↪ cleaned_text], images=image,
↪ return_tensors="pt", padding=True)
        inputs = {k: v.to(self.device) for k
↪ , v in inputs.items()}

        # Get features
        with torch.no_grad():
            outputs = self.clip_model(**
↪ inputs)
            logits_per_image = outputs.
↪ logits_per_image
            # Convert to similarity score
↪  (0-100 scale)
            clip_score = logits_per_image.
↪ squeeze().cpu().item()

        return float(clip_score)
    except Exception as e:
        print(f"Error calculating CLIP score
↪ : {e}")
        return 0.0

def generate_product_image_prompt(self,
↪ product_description):
    """Generate a complete image generation
↪ prompt from a product description."""
    # Set seed before translation for
↪ consistency
    torch.manual_seed(42)

    # Extract product attributes
    product_description = self.translator(
↪ product_description)[0]['translation_text'
↪ ]
    extracted_data = self.
↪ extract_product_attributes(
↪ product_description)

    # Format each element properly
    product_type = extracted_data.get('
↪ product_type', 'product')
```

```
        # Handle key_visual_features - could be
    ↪ string, list, or dict
        key_features = extracted_data.get('
    ↪ key_visual_features', 'distinctive
    ↪ features')
        if isinstance(key_features, list):
            key_features = ", ".join(
    ↪ key_features)
        elif isinstance(key_features, dict):
            key_features = ", ".join([f"{k}: {v}
    ↪ " for k, v in key_features.items()])

        # Handle color_material_details - could
    ↪ be string, list, or dict
        color_material = extracted_data.get('
    ↪ color_material_details', '')
        print(color_material)
        if isinstance(color_material, dict):
            color_material = ", ".join([f"{v}"
    ↪ for k, v in color_material.items()])
        elif isinstance(color_material, list):
            color_material = ", ".join(
    ↪ color_material)

        # Handle design_aspects - could be
    ↪ string, list, or dict
        design = extracted_data.get('
    ↪ design_aspects', '')
        if isinstance(design, list):
            design = ", ".join(design)
        elif isinstance(design, dict):
            design = ", ".join([f"{v}" for k, v
    ↪ in design.items()])

        # Build the enhanced prompt for CLIP
    ↪ scoring (more descriptive but still
    ↪ concise)
        simple_prompt = f"Product image of a {
    ↪ product_type}"
        # simple_prompt = f"{product_type} with
    ↪ {key_features}, {color_material} colors, {
    ↪ design} design"
        simple_prompt = re.sub(r'\s+', ' ',
    ↪ simple_prompt)
        simple_prompt = re.sub(r'\s,', ',',
    ↪ simple_prompt)
        simple_prompt = re.sub(r',,', ',',
    ↪ simple_prompt)

        # Build the detailed final prompt for
    ↪ image generation
        final_prompt = f"""Professional product
    ↪ photography of a {product_type},
featuring {key_features}, {color_material}, {
    ↪ design}, placed on a clean white
    ↪ background, laid flat,
studio lighting, 8k, 85mm lens f/2.8, no humans,
    ↪  no mannequins, no zoom, no tilt, no
    ↪ rotation, perfectly centered"""

        # Clean up any redundant spaces or line
    ↪ breaks
        final_prompt = re.sub(r'\s+', ' ',
    ↪ final_prompt)
        final_prompt = re.sub(r'\s,', ',',
    ↪ final_prompt)
        final_prompt = re.sub(r',,', ',',
    ↪ final_prompt)

        return final_prompt.strip(),
    ↪ simple_prompt.strip()
```

```
if __name__ == "__main__":
    # Set all random seeds for reproducibility
    SEED = 42
    random.seed(SEED)
    np.random.seed(SEED)
    torch.manual_seed(SEED)
    torch.cuda.manual_seed(SEED)
    torch.cuda.manual_seed_all(SEED)

    # Additional settings for reproducibility
    torch.backends.cudnn.deterministic = True
    torch.backends.cudnn.benchmark = False

    save_path = 'output'
    model_name = "microsoft/phi-3-mini-4k-
  ↪ instruct"
    generator = ProductPromptGenerator(
  ↪ model_name=model_name, device='cuda:0')
    g = torch.Generator(device='cuda:0')

    pipe = DiffusionPipeline.from_pretrained("
  ↪ black-forest-labs/FLUX.1-schnell",
  ↪ torch_dtype=torch.float16, device_map="
  ↪ balanced")
    pipe.load_lora_weights("aihpi/flux-fashion-
  ↪ lora")
    df = pd.read_excel('article_descriptions_3.
  ↪ xlsx', usecols=[0, 1, 2])

    # Initialize results storage
    clip_scores = []

    print("Starting image generation and CLIP
  ↪ scoring...")
    print("="*100)

    # Set generator seed once for all
  ↪ generations
    g.manual_seed(SEED)

    for idx in range(len(df)):
        print(f"Processing sample {idx + 1}/{len
  ↪ (df)}")

        # Get original description
        description = df.iloc[idx, :].tolist()
        original_description = '\n'.join([str(
  ↪ item) for item in description if pd.notna(
  ↪ item)])

        # Generate prompts (no need to reseed
  ↪ here as we want consistency within run)
        detailed_prompt, simple_prompt =
  ↪ generator.generate_product_image_prompt(
  ↪ original_description)
        print(f"Generated prompt: {
  ↪ detailed_prompt}")
        print(f"CLIP prompt: {simple_prompt}")

        # Generate image using detailed prompt
  ↪ with seeded generator
        image = pipe(detailed_prompt, generator=
  ↪ g).images[0]

        # Save image
        os.makedirs(save_path, exist_ok=True)
        image_path = f"{save_path}/{idx}.png"
        image.save(image_path)

        # Calculate CLIP score using simple
  ↪ prompt
        clip_score = generator.
  ↪ calculate_clip_score(image, simple_prompt)
        clip_scores.append(clip_score)
        print(f"CLIP Score: {clip_score:.4f}")

        # Save individual JSON for this sample
        result_entry = {
            "sample_id": idx,
            "original_description":
   ↪ original_description,
            "generation_prompt": detailed_prompt
   ↪ ,
            "clip_prompt": simple_prompt,
            "clip_score": clip_score,
            "image_path": image_path
        }

        json_path = f"{save_path}/{idx}.json"
        with open(json_path, "w", encoding="utf
   ↪ -8") as f:
            json.dump(result_entry, f, indent=2,
   ↪  ensure_ascii=False)

        print('-'*100)

    # Calculate and save statistics
    total_samples = len(clip_scores)
    mean_score = statistics.mean(clip_scores)
    median_score = statistics.median(clip_scores
   ↪ )
    std_dev = statistics.stdev(clip_scores) if
   ↪ len(clip_scores) > 1 else 0.0
    min_score = min(clip_scores)
    max_score = max(clip_scores)

    # Simple statistics output
    stats_text = f"""CLIP Score Statistics:
Total Samples: {total_samples}
Mean: {mean_score:.4f}
Median: {median_score:.4f}
Std Dev: {std_dev:.4f}
Min: {min_score:.4f}
Max: {max_score:.4f}
"""

    # Print to console
    print("\n" + "="*30)
    print("CLIP SCORE SUMMARY")
    print("="*30)
    print(stats_text)

    # Save to text file
    with open(f"{save_path}/clip_statistics.txt"
   ↪ , "w", encoding="utf-8") as f:
        f.write(stats_text)

    print(f"Individual JSON files saved in:
   ↪ newrun1/")
    print(f"Statistics saved to: clip_statistics
   ↪ .txt")
```