



POLYTECHNIC UNIVERSITY OF BARI

DEPARTMENT OF ELECTRICAL AND INFORMATION ENGINEERING
MASTER'S DEGREE COURSE IN COMPUTER ENGINEERING
ARTIFICIAL INTELLIGENCE AND DATA SCIENCE

Deep Learning – Project 2025_ VI

PikaPikaGen

Generative Synthesis of Pokémon
Sprites from Textual Descriptions

Student:

Luigi Battista

ID: 590703

Teacher:

Vito Walter Anelli

July 2, 2025

Academic Year: 2024/2025

Contents

1	Introduction	1
2	Dataset	2
2.1	Image Collection	2
2.2	Image Segmentation	4
2.3	Description Curation	5
2.4	Paraphrasing	6
2.5	Dataset Splitting and Preprocessing	8
3	Model	10
3.1	Text Encoder	11
3.1.1	BERT-Mini Text Encoder	12
3.1.2	CLIP Text Encoder	13
3.1.3	Fine-Tuning Strategy	15
3.2	Generator	16
3.3	Discriminator	18
3.4	Loss function	19
3.4.1	Discriminator Loss	19
3.4.2	Generator Loss	21
4	Results	22
4.1	Training Details	22
4.2	Evaluation Metrics	24
4.2.1	Inception Score (IS)	24
4.2.2	Fréchet Inception Distance (FID)	25
4.2.3	Kernel Inception Distance (KID)	26
4.2.4	CLIP Score	27
4.3	Quantitative and Qualitative Evaluation	28
4.3.1	Quantitative Results	28
4.3.2	Qualitative Evaluation	29
5	Conclusion	34
	Bibliography	36

1 Introduction

Text-to-image synthesis is a complex multimodal task that involves generating coherent visual content from natural language descriptions. It requires a deep semantic understanding of textual input, the ability to model visual structures and an effective mechanism for aligning concepts across linguistic and visual modalities.

In this project, the goal was to design a model capable of synthesizing Pokémon-style sprite images from their corresponding Pokédex descriptions. These descriptions typically convey visual traits, elemental types and behavioral characteristics, serving as high-level semantic guides for image generation. The assignment specified the use of an Encoder–Decoder architecture with an Attention mechanism to bridge the textual and visual modalities.

To address this, a conditional *Generative Adversarial Network* (cGAN) framework was adopted. Although the cGAN architecture includes both a generator and a discriminator during training, only the generator is retained at inference time. As such, the generator effectively functions as a decoder conditioned on the encoded text, thereby fulfilling the Encoder–Decoder design outlined in the assignment.

A critical challenge encountered was the limited size and structure of the provided dataset. This contains a limited number of examples – approximately 900 entries – each consisting of a sprite and a short textual description. While it offered a valuable starting point it was insufficient in both scale and diversity for training a data-hungry generative model. To address this, a custom dataset was constructed, incorporating paraphrased descriptions and additional images for each Pokémon, to modestly expand the training set. Even with these efforts, the dataset remains small by the standards of generative modeling, and this inevitably constrained the model’s learning capacity.

Nonetheless, the model demonstrated the ability to generate visually plausible and stylistically consistent Pokémon sprites. It captured coarse semantic features – such as general shape, color, or notable attributes – but finer-grained alignment with the input descriptions remained limited. This outcome reflects both the inherent difficulty of the task and the dataset’s constrained size.

The training process remained stable throughout, and common GAN pathologies such as mode collapse or divergence were successfully avoided.

2 Dataset

The provided dataset consists of a CSV file containing structured Pokémon information, paired with a corresponding image set. The CSV includes various attributes such as the Pokémon’s name, primary and secondary types, numerical stats (e.g., HP, Attack, Defense), effectiveness against other types, evolution information and a short textual description typically taken from trading cards. However, this description is often too simplistic or incomplete to serve as a rich visual reference for generative tasks. It generally lack crucial details about color, shape, body structure and distinctive visual elements that are essential for guiding a text-to-image model. Additionally, the accompanying image set, consisting of a **single** high-quality segmented picture per Pokémon, is insufficient to support robust generative modeling.

Given these limitations, I opted to build a custom dataset from scratch, tailored specifically for the requirements of text-to-image generation. The objective was to create a more diverse and visually rich dataset that captures both the **visual diversity** of Pokémon characters and the **linguistic variation** in how they can be described. This required a multi-phase pipeline, which I designed and implemented as follows:

1. **Image Collection** – Gathering a diverse set of anime-style images for each Pokémon from online sources.
2. **Image Segmentation** – Isolation of characters from backgrounds to help the model focus solely on the character’s visual features.
3. **Description Curation** – Generation of visually grounded, descriptive captions for each character.
4. **Paraphrasing** – Generating alternative phrasings for each description to improve the model’s robustness and generalization.

Each of these stages is detailed in the sections below, describing the motivation, methods and tools involved in building a such dataset.

2.1 Image Collection

The first step in building the dataset involved assembling a high-quality, diverse image collection of Pokémon characters. Given the lack of any sufficiently varied dataset online tailored to generative modeling, I chose to construct a **custom dataset** from scratch by automating the image retrieval process. I focused primarily on first-generation Pokémon, as they are the most widely recognized and have the largest volume of available visual material online. In total, I considered 115 different Pokémon.

To gather the images, I used *icrawler*¹, a Python library that enables automated scraping of image data from major search engines. Leveraging its support for both *Google Images* and *Bing*, I was able to significantly increase the diversity and number of illustrations collected for each Pokémon. To improve the relevance and quality of the images retrieved, I defined a set of search queries designed to target the anime-style appearance of the character. These queries combined the Pokémon’s name with additional keywords such as «official anime art», «anime screenshot» and «anime official». For each query on each search engine, the crawler was set to retrieve up to 50 images, totaling a theoretical maximum of 300 images per Pokémon (3 queries \times 2 engines \times 50 images). However, in practice, the number of successfully downloaded images was often lower due to occasional download failures or errors from the underlying library. The retrieved images were organized in a separate folder for each Pokémon, making easier to process them in subsequent steps.

Because many images retrieved from different search engines were visually identical or slightly altered versions of the same picture, I created an automated process to remove duplicates. I used the *phash* function from the *imagehash*² library, which creates a compact fingerprint of an image based on how it looks, rather than its exact file data. This method can detect duplicates even when images have been resized, compressed or slightly edited. For each new image, a *perceptual hash* was generated and compared to those already seen. If the hash matched one in the existing set, the image was considered a duplicate and discarded.

This automated pipeline successfully collected and curated thousands of Pokémon images, minimizing manual intervention while eliminating redundant or near-identical samples.

Despite the focused search strategy, many irrelevant images still slipped through. Common unwanted cases included:

- Product and merchandise photos
- Pixel art, game sprites or low-resolution assets
- Images showing evolutionary forms of the searched Pokémon
- Fan art with inconsistent visual styles

As a result, a manual filtering step was required to remove these undesired samples and ensure the dataset aligned with the intended visual style.

¹<https://github.com/hellock/icrawler>

²<https://github.com/JohannesBuchner/imagehash>

2.2 Image Segmentation

Following the image collection phase, the next phase focused on **segmenting the Pokémon characters from their backgrounds**. The objective was to isolate each character to help the text-to-image model focus purely on the visual characteristics of each Pokémon, minimizing distractions caused by complex or inconsistent backgrounds.

As an initial approach, I searched for top-performing models on Hugging Face, specifically for the *background removal task*³, sorting them by popularity and user ratings to identify the most promising candidates for object segmentation. I tested several of the top-ranked models, including *RMBG-1.4*⁴ and *BiRefNet*⁵. However, the results were often unsatisfactory – many models failed to accurately isolate the Pokémon characters, frequently omitting important body parts or incorrectly segmenting portions of the background instead.

To achieve more reliable and accurate results, I turned to the ***Grounded-Segment-Anything (GSA)***⁶ [16], a state-of-art framework that combines two relevant visual models:

- **Grounding DINO** [12]: Performs object detection based on natural language prompts in a zero-shot manner by identifying bounding boxes around relevant regions in the image.
- **Segment Anything Model (SAM)** [10]: Takes these bounding boxes and generates high-quality segmentation masks from them.

This combination allows for segmenting arbitrary objects in an image using only a textual prompt.

I applied this pipeline to each Pokémon image in the curated dataset. If an image already included an alpha (transparency) channel – indicating that the background had been previously removed – it was preserved without further processing. Otherwise, the image was passed through the GSA pipeline using the prompt «Pokémon character».

Grounding DINO first identified regions in the image that matched the prompt and produced bounding boxes around the most likely candidates. For each image, up to the **top-5 bounding boxes** (ranked by confidence score) were passed to SAM, which generated segmentation masks for these regions. Each mask was then used to generate a new version of the image with a transparent background.

³<https://huggingface.co/models?other=background-removal>

⁴<https://huggingface.co/briaai/RMBG-1.4>

⁵<https://huggingface.co/ZhengPeng7/BiRefNet>

⁶<https://github.com/IDEA-Research/Grounded-Segment-Anything>

Since the automated process wasn’t always perfect, multiple segmentation outputs were retained for each image to improve the chances of capturing a clean and complete character. These candidate outputs were manually reviewed, and the best one – the one that most accurately captured the Pokémon without excess background or missing parts – was selected for the final dataset.

In most cases, the GSA pipeline produced accurate and reliable results. However, for a minority of images where the segmentation was poor or the character was partially cropped, I performed manual refinements using an online background removal tool to ensure consistent quality.

After this semi-automated process and manual quality control, the final dataset consisted of approximately **40 high-quality, background-free images** per Pokémon. Representative samples illustrating the variability of the images for each Pokémon are presented in Figure 2.1.



Figure 2.1: Representative samples of segmented, background-free images for *Charmander* (top row) and *Bulbasaur* (bottom row).

Despite substantial efforts to augment both the size and fidelity of the image dataset, its overall scale remains relatively small compared to the volumes typically required for robust text-to-image synthesis. This limitation inevitably poses challenges in achieving high diversity and fine-grained semantic alignment in the generated images.

2.3 Description Curation

To accompany the visual dataset and guide the text-to-image generation model, I curated a set of **concise, visually-focused descriptions** for each Pokémon. As a starting point, I used the *Pokémon Dataset 2024*⁷ on Kaggle, which includes

⁷<https://www.kaggle.com/datasets/ceebloop/pokmon-dataset-2024>

comprehensive textual entries for all Pokémon compiled from *Bulbapedia*⁸, a fan-maintained Pokémon encyclopedia that offers detailed entries on each character.

These entries typically include background lore, evolutionary lines, battle attributes, typing and visual traits. For example, the *Bulbasaur*⁹ page covers not only its appearance but also information about its behavior, type, and evolution. While informative, much of this content is not useful for the training objective.

To isolate the relevant visual content, I used **ChatGPT (GPT-4o)**¹⁰ to re-process each entry and extract only the essential visual characteristics, following a strict set of guidelines. Each description was required to:

- Include shape, size, primary colors, body structure (e.g., limbs, eyes, ears, tails) and any notable visual patterns or features.
- Be limited to 40–60 words to ensure consistency across samples.

Descriptions explicitly excluded:

- Evolutionary details or historical background
- Gender differences or alternate forms
- Typing, battle roles or categorization
- Any use of the word “Pokémon”

The result was a structured CSV file pairing each character with a refined, visually descriptive caption, designed to align with the model’s learning objectives.

2.4 Paraphrasing

To enhance the model’s ability to generalize across variations in input phrasing, I introduced a **paraphrasing step**. This aimed to generate semantically equivalent descriptions that differed in wording or structure. The goal was twofold:

- Improve robustness during training by exposing the model to alternate phrasings.
- Use paraphrased descriptions during testing to evaluate the model’s capacity to generalize beyond the exact texts it was trained on.

⁸https://bulbapedia.bulbagarden.net/wiki/Main_Page

⁹<https://bulbapedia.bulbagarden.net/wiki/Bulbasaur>

¹⁰<https://chat.openai.com/>

I searched for top-performing models on Hugging Face specifically for the *paraphrase task*¹¹ and I evaluated the top-3 of them:

- *pegasus_paraphrase*¹²
- *parrot_paraphraser_on_T5*¹³
- *chatgpt_paraphraser_on_T5_base*¹⁴

After comparison, the model *chatgpt_paraphraser_on_T5_base* emerged as the most suitable. It consistently generated high-quality paraphrases that retained the core meaning while varying sentence structure and vocabulary.

Using this model, I generated **two paraphrases** for each original description:

- One used as a **training variant**, enriching the input space.
- One reserved for **testing**, providing unseen input to evaluate model generalization.

The paraphrasing was implemented using a diverse decoding setup with *beam search*, repetition and diversity penalties and *n-gram blocking* to maximize output quality and uniqueness. Only paraphrases that were distinct from the original were retained.

The final output was a new CSV file containing:

- **name**: Pokémon name
- **description**: Main visual description
- **train_paraphrase**: Used alongside the original for training
- **test_paraphrase**: Used for testing generalization

This text dataset became the conditioning component of the image generation pipeline, enabling the training of a model that is not only grounded in detailed visual language but also resilient to natural linguistic variability.

¹¹<https://huggingface.co/models?sort=likes&search=paraphrase>

¹²https://huggingface.co/tuner007/pegasus_paraphrase

¹³https://huggingface.co/prithivida/parrot_paraphraser_on_T5

¹⁴https://huggingface.co/humarin/chatgpt_paraphraser_on_T5_base

2.5 Dataset Splitting and Preprocessing

To structure the dataset for effective training and evaluation, the full set of curated images was divided into *training* (90%), *validation* (3%) and *intra-class test* (7%) subsets, with stratification across the 115 first-generation Pokémon. To assess the model’s generalization capabilities, I additionally created a ***novel-class test*** set comprising 402 images from Pokémon not included in the original 115, sampled from the original assignment dataset.

To improve loading efficiency and consistency, all images were converted to RGB format with a white background, removing alpha transparency. All processed images were saved in `.jpg` format using high-quality compression. This pre-processing step helped reduce computational overhead during training. Final image dataset sizes were:

- **Train:** 4022 images
- **Validation:** 101 images
- **Intra-Test:** 403 images
- **Novel-Test:** 402 images

Given the sensitivity of text-to-image models to color fidelity and spatial structure, **data augmentation** was applied conservatively. The augmentation pipeline included:

- Random horizontal flipping
- Mild affine transformations (rotation, scaling, translation)
- Subtle color jitter (brightness, contrast, saturation, hue)

All transformations were applied over a consistent white background to maintain visual coherence. In addition to these, all images were resized to a fixed resolution and normalized in the range of $[-1, 1]$.

Since generative models are highly sensitive to color shifts and distortions, augmentation parameters were carefully tuned to preserve the semantic and visual integrity of the Pokémon characters, while introducing sufficient variability to prevent overfitting. Augmentations were applied exclusively to the training set; validation and test sets were processed using only resizing and normalization.

Finally, text prompts were sampled for each image to introduce controlled linguistic variation during training. Each training (and validation) sample had an 80% chance of using the original visual description and a 20% chance of using a

paraphrased variant (`train_paraphrase`). This encouraged the model to handle varied linguistic phrasing while maintaining grounding in consistent visual content.

For **intra-class testing**, only `test_paraphrase` entries – unseen during training – were used, while for **novel-class testing** it was possible to use directly the original description given that these refer to Pokémon never seen during training.

3 Model

The proposed model builds upon the **conditional GAN (cGAN)** [13] framework, with architectural inspiration drawn from *DF-GAN* [22], a strong baseline in text-to-image generation. However, the text conditioning mechanism is substantially reworked to improve semantic expressivity and alignment.

DF-GAN adopts a *Bi-directional LSTM* [19] as its text encoder – specifically reusing the pretrained encoder from *AttnGAN* [24] – to obtain a global sentence embedding. This embedding is injected into the generator through repeated affine transformations, where per-channel scaling and shifting parameters are learned from the sentence embedding and applied across multiple spatial scales via a *Deep Fusion Block* (DFBlock) [22]. While effective, this method relies solely on a compressed sentence-level representation and uses deterministic modulation functions for fusion, limiting the model’s ability to condition on finer-grained textual elements.

In contrast, the proposed architecture employs the **CLIP text encoder** [15], which yields both a *global sentence embedding* and *per-token contextual embeddings*. This richer textual representation enables a more expressive and spatially-aware conditioning strategy. Specifically, conditioning in the generator is achieved through two complementary pathways:

- **Global Conditioning:** The sentence embedding is concatenated with the latent noise vector at the input layer, ensuring the generator receives high-level semantic context from the outset.
- **Local Conditioning:** Token-level embeddings are injected via *cross-attention* layers distributed across multiple stages of the generator, allowing fine-grained semantic alignment throughout the generation process.

An overview of the complete architecture is provided in Figure 3.1.

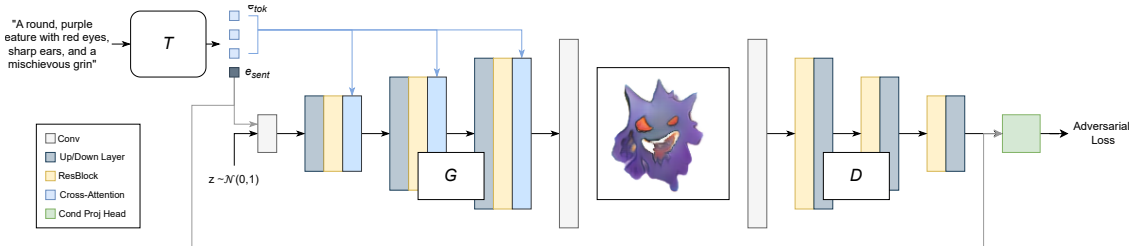


Figure 3.1: Overall architecture of the proposed model.

First, the CLIP text encoder T processes each input description and produces two types of representations: a global sentence embedding e_{sent} and a sequence of contextualized token embeddings e_{tok} .

The generator G receives as input a latent noise vector z sampled from a Gaussian distribution, concatenated with the sentence embedding e_{sent} . This input is

passed through a sequence of structurally identical blocks that progressively up-sample the feature map and refine visual detail. Each block consists of: (i) a learnable upsampling layer, (ii) a residual convolutional block and (iii) a cross-attention module that injects token-level semantic features e_{tok} . The final image is generated through a convolutional head followed by a \tanh activation, mapping the internal feature representation to the RGB domain.

The discriminator D follows a symmetric design, processessing the generated (or real) image through a sequence of downsampling residual blocks that reduce spatial resolution while deepening feature abstraction. At the final stage, a *conditional projection head* combines the aggregated visual features with the corresponding sentence embedding e_{sent} to produce the discriminator’s scalar output, assessing both realism and semantic alignment.

The CLIP encoder is lightly fine-tuned to better adapt its representations to the Pokémon domain. In contrast, both the generator and discriminator are trained from scratch. The motivations behind adopting CLIP and the details of its fine-tuning are discussed in the following section. Subsequent sections also provide an in-depth description of the generator and discriminator components.

3.1 Text Encoder

To condition the image generation process on textual prompts, a flexible text encoding module was implemented. This module supports two alternative pretrained language models: **BERT-Mini**¹ and the **CLIP ViT-B/32 text encoder**². While the original assignment specified the use of BERT-Mini due to resource constraints, the integration of CLIP was additionally explored to assess the impact of vision-aligned text embeddings, which have demonstrated superior performance in capturing multimodal correspondences.

The encoder module takes raw textual inputs and produces two forms of output:

- A sequence of contextualized token embeddings of shape $(T \times D)$, which is used as input to the generator’s cross-attention mechanism.
- A global sentence embedding of shape $(1 \times D)$, derived from special tokens ([CLS] or [EOS]), which is used for global conditioning in both the generator and the discriminator.

Since CLIP’s text encoder operates in a 512-dimensional space, an additional linear projection layer was appended to map its output to a shared 256-dimensional latent

¹<https://huggingface.co/prajjwal1/bert-mini>

²<https://huggingface.co/openai/clip-vit-base-patch32>

space, consistent with the rest of the generative architecture. In contrast, BERT-Mini natively outputs 256-dimensional embeddings, and therefore no dimensionality reduction was required.

Both encoders were fine-tuned using a partial adaptation strategy to ensure alignment with the specific linguistic distribution of the downstream task while avoiding overfitting. Detailed architectural and fine-tuning specifications for each encoder are provided in the following sections. While both encoders were implemented and evaluated, CLIP’s vision-aligned embeddings were ultimately employed in the final system due to its better alignment with visual semantics.

3.1.1 BERT-Mini Text Encoder

BERT-Mini is a lightweight version of the standard BERT model [3], designed to retain essential representational capacity, while reducing computational overhead. It follows the standard BERT framework, but with fewer layers and smaller hidden dimensions. The model is pretrained using the **Masked Language Modeling (MLM)** objective, where random tokens in the input are masked and the model learns to predict them by leveraging bidirectional context from both left and right. This enables the model to learn rich, deep, and bidirectional dependencies across large-scale unlabeled text corpora.

Input text is processed by the *BERT tokenizer*, which applies **WordPiece tokenization** [23] and maps each token to a fixed vocabulary of 30,522 entries. The tokenizer inserts special tokens such as [CLS] (used for pooled sentence-level representation) and [SEP] (used to denote segment boundaries) and pads all sequences to a uniform maximum length.

The tokenized input is embedded through the sum of three learnable components: *word embedding layer*, which map each token in the vocabulary (size 30,522) to a 256-dimensional vector; *token type embedding layer*, indicating segment membership (e.g., sentence A vs. sentence B in paired inputs); and *positional embeddings*, representing learned absolute position encodings for up to 512 input positions. In this setting, where all inputs are single sequences, token type embeddings remain constant but are still included in the summation.

This composite embedding undergoes *layer normalization* ($\epsilon = 1\text{e-}12$) and *dropout* regularization ($p = 0.1$) before being fed into the encoder stack.

The Transformer encoder consists of **4 layers**, each comprising two main sub-components: a *Multi-Head Self-Attention* (MHSA) and a *Feed-Forward Network* (FFN). Each MHSA employs 4 attention heads, each with a head dimension of 64, resulting in a total hidden size of 256. Scaled dot-product attention is computed independently in each head and then concatenated and projected back into the original 256-dimensional space. This mechanism allows the model to attend to information at multiple representational subspaces simultaneously.

The FFN within each layer consists of a two-layer fully connected network with

a hidden expansion to 1024 dimensions, followed by a GELU activation, and then a projection back to 256 dimensions.

Both MHSA and FFN sublayers are enclosed in residual connections and followed by *Layer Normalization* ($\epsilon = 1\text{e-}12$) to stabilize training and improve gradient flow. *Dropout* ($p = 0.1$) is applied after both sublayers to enhance regularization.

The model outputs two types of representations: *contextualized token embeddings*, encoding the position-aware semantic content of each input token; and a *sentence embedding*, obtained by applying a linear projection and tanh activation to the [CLS] token, representing a fixed-length summary vector of the entire input sequence.

Full architectural parameters are detailed in Table 3.1.

Table 3.1: BERT-Mini text encoder architecture summary

Component	Configuration
Tokenizer	WordPiece
Maximum Input Length	512
Vocabulary Size	30,522
Embedding Dimension	256
Transformer Layers	4
Number of Attention Heads	4
Head Dimension	64
FFN Hidden Dimension	1024
Activation Function	GELU
Normalization	LayerNorm
Output Dimensions	Token: (512×256) , Sentence: (1×256)

3.1.2 CLIP Text Encoder

CLIP (Contrastive Language–Image Pretraining) [15] is a multimodal framework developed to jointly learn aligned representations of images and text. It is pretrained on a massive dataset of 400 million image-text pairs using a **contrastive learning objective**, which learns to maximize the similarity between matching image-text pairs while minimizing it for non-matching pairs. This encourages the model to produce embeddings where matching image and text pairs are close in the shared latent space, while non-matching pairs are pushed apart. CLIP is composed of two modality-specific encoders – a vision encoder and a text encoder. In this work, we leverage **only the text encoder** component.

Input text is first processed by the *CLIP tokenizer*, which applies *Byte-Pair Encoding (BPE)* [20] to segment text into subword units and maps each token

to a fixed vocabulary of 49,408 entries. The tokenizer prepends a special start-of-sequence [SOS] token and appends an end-of-sequence [EOS] token to each input. Sequences longer than the model’s maximum length (77 tokens) are truncated, and shorter ones are padded to maintain uniform input size.

Tokens are embedded via a learnable token embedding layer, which maps each token in the vocabulary (size 49,408) to a 512-dimensional vector. Positional information is incorporated through a learned absolute positional embedding, which is added element-wise to the token embeddings. Unlike BERT, CLIP does not utilize segment embeddings. Therefore, the sum of token and positional embeddings forms the input to the Transformer encoder stack.

The Transformer encoder comprises **12 layers**, each consisting of a MHSA and FFN sublayers. Each MHSA uses 8 attention heads with a head dimension of 64, yielding a total embedding dimension of 512. The FFN expands the hidden size to 2048, followed by the QuickGELU activation function – an approximation of GELU designed for faster computation, while maintaining smooth nonlinearities – and then projects back to 512 dimensions.

Residual connections and layer normalization (with $\epsilon = 1e-5$) are applied after both the sublayers to promote stable training and gradient flow. Dropout is also applied for regularization. A final layer normalization is applied after the last Transformer layer.

At the output, the model produces *contextualized token embeddings* for each token in the input sequence and a *global sentence embedding*, obtained by extracting the embedding of the EOS token and passing it through a linear projection layer.

Full architectural parameters are detailed in Table 3.2.

Table 3.2: CLIP (ViT-B/32) text encoder architecture summary

Component	Configuration
Tokenizer	Byte-Pair Encoding (BPE)
Maximum Input Length	77
Vocabulary Size	49,408
Embedding Dimension	512
Transformer Layers	12
Number of Attention Heads	8
Head Dimension	64
FFN Hidden Dimension	2048
Activation Function	QuickGELU
Normalization	LayerNorm
Output Dimensions	Token: (77×512) , Sentence: (1×512)

3.1.3 Fine-Tuning Strategy

Given the limited size of the task-specific dataset and the high capacity of pre-trained models such as *CLIP* and *BERT-Mini*, a full fine-tuning approach posed a considerable risk of overfitting and loss of generalization. To mitigate this, a **partial fine-tuning strategy** was employed, where the majority of model parameters were kept frozen and updates were restricted to a carefully selected subset of components. This strategy enables modest task-specific adaptation while preserving the general-purpose knowledge encoded in the pre-trained representations.

The following components were selectively fine-tuned:

- **Embedding Layers:** While the bulk of the model’s parameters were frozen, the word/token embedding layers were unfrozen to allow adaptation of the input representation to the domain-specific vocabulary. Fine-tuning these layers permits the model to refine how input tokens are grounded in vector space, which is especially important when the target corpus includes specialized terminology or exhibits different usage patterns than the pretraining corpus. Importantly, the positional embeddings were left frozen, maintaining the learned structural priors.
- **Final Transformer Layer:** The last Transformer block was unfrozen to allow limited structural adaptation of the contextualized representations. Fine-tuning only the upper layer of the encoder permits the model to reshape the final feature space while preserving the integrity of the lower-level representations learned during pretraining. This has been shown empirically to yield good transfer performance, especially in low-data regimes.
- **Layer Normalization Parameters:** All LayerNorm affine parameters (i.e., scale γ and shift β) were fine-tuned throughout the model. This design choice is motivated by prior findings indicating that adapting normalization statistics can lead to significant performance gains even when the rest of the network remains frozen. These parameters control the internal feature scaling and distribution across layers and thus provide a lightweight mechanism for tuning activations to the characteristics of the downstream task.
- **Sentence Embedding Projection Layers:** Both CLIP and BERT-Mini include pretrained layers that project internal representations to fixed-length sentence embeddings. In BERT-Mini, it is the **pooler** layer applied to the [CLS] token embedding, while in CLIP, this corresponds to the **text_projection** linear layer applied to the final [EOS] token embedding. These pretrained projection heads were kept trainable, enabling refined alignment of global sentence-level features without modifying the rest of the encoder stack.

This fine-tuning scheme allowed moderate adaptation without destabilizing pre-trained representations. The additional projection layer introduced to align CLIP’s

output to 256 dimensions was, instead, trained from scratch and learned jointly with the rest of the model.

3.2 Generator

The generator architecture is designed to progressively synthesize high-resolution images from a low-dimensional noise vector, conditioned on both sentence and per-token embeddings. It integrates the sentence embedding by concatenating it with the latent noise vector at the input stage, ensuring that the generator is exposed to high-level semantic context from the very beginning. In contrast, per-token embedding is injected via cross-attention modules distributed across multiple spatial resolutions.

The overall architecture consists of a sequence of identical groups that gradually refine the feature representation while allowing fine-grained semantic control over the generation process. Each group is composed of: (i) an upsampling layer, (ii) a residual block and (iii) a cross-attention block. The final image is produced via a convolutional head followed by a \tanh activation function.

The architecture is resolution-adaptive. For a 128×128 output image, the detailed configuration is summarized in Table 3.3.

Table 3.3: Generator architecture configuration for 128×128 output resolution.

Group	Kernel Size	Stride	Padding	Output Channels	Resolution
Initial Projection	—	—	—	256	4×4
Upsample 1	3×3	2×2	1	256	8×8
Residual Block 1	3×3	1×1	1	256	8×8
Cross-Attention 1	—	—	—	256	8×8
Upsample 2	3×3	2×2	1	256	16×16
Residual Block 2	3×3	1×1	1	256	16×16
Cross-Attention 2	—	—	—	256	16×16
Upsample 3	3×3	2×2	1	128	32×32
Residual Block 3	3×3	1×1	1	128	32×32
Cross-Attention 3	—	—	—	128	32×32
Upsample 4	3×3	2×2	1	64	64×64
Residual Block 4	3×3	1×1	1	64	64×64
Cross-Attention 4	—	—	—	64	64×64
Upsample 5	3×3	2×2	1	32	128×128
Residual Block 5	3×3	1×1	1	32	128×128
Cross-Attention 5	—	—	—	32	128×128
Final RGB Head	3×3	1×1	1	3	128×128

Initial Projection. The input to the generator consists of a latent noise vector $z \in \mathbb{R}^{128}$ sampled from Gaussian distribution and a sentence embedding $e_{\text{sent}} \in \mathbb{R}^{256}$.

These are concatenated and projected via a linear layer into a 4×4 spatial feature map with 256 channels.

Upsampling Layer. To increase spatial resolution, each group begins with an upsampling layer. By default, *bilinear interpolation* followed by a 3×3 convolution is used. This design choice avoids the checkerboard artifacts commonly associated with transposed convolutions, thereby improving visual fidelity. Nevertheless, transposed convolutions (`ConvTranspose2d`) are also supported and can be activated via a configuration flag, in compliance with assignment requirements. At each stage, the resolution is doubled, progressing from 4×4 up to 128×128 .

Residual Block. Each upsampled feature map is processed by a residual block inspired by ResNet architectures [5]. These blocks consist of two 3×3 convolutional layers, each followed by *Group Normalization* (32 groups) and *Sigmoid Linear Unit* (SiLU) activations. A skip connection links the input to the output, either directly or via a 1×1 convolution when the number of channels differs. Residual connections were adopted for their ability to maintain stable gradient flow, which is essential for training deep models effectively. This helps preserve information flow across layers, enhancing output quality and diversity. I opted to employ Group Normalization with 32 groups instead of the conventional Batch Normalization due to its independence from batch size, which ensures more consistent and stable gradient estimates, particularly beneficial when training with small or variable batch sizes. Additionally, the SiLU activation function was chosen over ReLU for its smoother non-linearity, which facilitates more effective gradient propagation and has demonstrated empirical performance improvements in recent vision architectures, notably in models such as Stable Diffusion [17].

Cross-Attention Block. Following the residual block, each group includes a Cross-Attention Block, which injects per-token embeddings at each resolution stage; here, spatial image features serve as queries, with token embeddings providing keys and values. Multi-head attention is employed with 4 heads and a head dimension of 64, resulting in a total hidden dimension of 256. Query and key vectors are L2-normalized for cosine-similarity attention, ensuring consistent scaling, while value vectors remain unnormalized. For enhanced training speed and memory efficiency, attention is computed using PyTorch’s `scaled_dot_product_attention` (SDPA), which fuses operations and enables optimized kernel selection. Finally, the attention output undergoes a linear projection to map it back to the spatial feature space, enabling seamless integration with subsequent convolutional layers.

Final RGB Head. Following the last group, a final 3×3 convolutional layer maps the feature map to a 3-channel RGB image. The output is normalized with

GroupNorm, activated with SiLU and squashed to the $[-1, 1]$ normalized image range via a `tanh` non-linearity.

3.3 Discriminator

The discriminator aims to assess whether a generated image is realistic and semantically aligned with a conditioning text description. To this end, it processes input images through a hierarchy of downsampling residual blocks that progressively reduce spatial resolution while increasing feature dimensionality. At the end, a *conditional projection head* fuses the resulting feature map with the sentence embedding to predict the final discriminator’s confidence score.

As with the generator, the discriminator architecture is resolution-adaptive. Table 3.4 shows the configuration for input images of size 128×128 .

Table 3.4: Discriminator architecture configuration for 128×128 input resolution.

Block	Kernel Size	Stride	Padding	Output Channels	Resolution
Initial Conv	3×3	1×1	1	32	128×128
ResBlockD 1 + Down	3×3	2×2	1	64	64×64
ResBlockD 2 + Down	3×3	2×2	1	128	32×32
ResBlockD 3 + Down	3×3	2×2	1	256	16×16
ResBlockD 4 + Down	3×3	2×2	1	256	8×8
ResBlockD 5 + Down	3×3	2×2	1	256	4×4
Conditional Projection Head	$3 \times 3 + 4 \times 4$	1×1	1 / 0	1	1×1

Initial Projection The discriminator begins with a 3×3 convolutional layer that maps the 3-channel RGB image to a lower-dimensional feature space with 32 channels. This prepares the input for hierarchical downsampling via residual blocks.

Residual Blocks with Downsampling. The discriminator employs a stack of downsampling residual blocks that progressively reduce spatial resolution while increasing channel depth. Each block contains two 3×3 convolutional layers followed by LeakyReLU activations with a negative slope of 0.2. This variant allows a small gradient to flow for negative inputs, preventing neuron "dying" and helping maintain gradient flow throughout deep architectures. LeakyReLU is widely adopted in GAN discriminators due to its empirical success in preserving feature diversity and training stability. Instead of using BatchNorm or GroupNorm, the discriminator employs **Spectral Normalization** [14] on all convolutional layers. Spectral Norm constrains the Lipschitz constant of each layer by normalizing its weight matrix via its largest singular value. This ensures smoother gradients and prevents the discriminator from becoming too sharp or overconfident, which is known to destabilize GAN training. Furthermore, unlike conventional designs where downsampling is

applied *after* the residual block, here downsampling is embedded *within* the block itself and applied to both the main and skip paths. This design fosters more stable and effective feature learning at each resolution level, allowing the skip path to directly contribute information at the target downsampled scale and thereby improving overall gradient flow in deep networks. Downsampling is performed using either a stride convolution or a average pooling operation. The default method is strided convolution, which applies a 4×4 kernel with a stride of 2, allowing the model to learn the downsampling transformation. Alternatively, non-parametric 2×2 average pooling can be selected to reduce resolution in a fixed manner. The choice between these downsampling strategies is controlled via a configuration flag.

Conditional Head. After the final residual block, the resulting 4×4 feature map is fused with the sentence embedding to guide the final prediction. The sentence embedding $e_{\text{sent}} \in \mathbb{R}^{256}$ is first broadcast across the spatial dimensions (to match the feature map shape) and then concatenated along the channel axis with the image feature map. This fused representation is processed through two convolutional layers: a 3×3 convolution followed by LeakyReLU and a final 4×4 convolution that compresses the map to a scalar output. This scalar represents the discriminator’s confidence that the input image is both real and matches the given text.

3.4 Loss function

The model is trained using adversarial learning, in which a generator G and a discriminator D are optimized with opposing objectives in a minimax game [4]:

$$\min_G \max_D \mathcal{L}(G, D) \quad (3.1)$$

The generator learns to synthesize photorealistic images that are semantically aligned with input text descriptions, while the discriminator learns to distinguish between real and generated images and assess their consistency with the provided text. The loss formulation follows the design of DF-GAN [22], combining hinge adversarial loss terms with a Matching-Aware Gradient Penalty (MA-GP). Hinge losses are adopted given their strong empirical performance in stabilizing GAN training [11, 14]. An additional gradient penalty term is employed to improve training stability and enhance the discriminator’s ability to provide semantically meaningful feedback to the generator. The following sections present the loss functions for the discriminator and generator, which are both formulated as minimization problems suitable for optimization with gradient-based methods.

3.4.1 Discriminator Loss

During training, the discriminator observes three types of input pairs:

1. A real image with a matching text description,
2. A generated image conditioned on a matching description,
3. A real image with a mismatched description.

For each of these, a separate hinge loss is computed. The full discriminator objective is defined as:

$$\mathcal{L}_D = \mathcal{L}_{\text{real}} + \frac{1}{2}\mathcal{L}_{\text{fake}} + \frac{1}{2}\mathcal{L}_{\text{mis}} + \lambda_{\text{MA-GP}} \cdot \mathcal{L}_{\text{MA-GP}}, \quad (3.2)$$

where $\lambda_{\text{MA-GP}}$ is an hyperparameter to balance the effectiveness of gradient penalty.

Each component contributes to a specific aspect of the discriminator’s learning objective:

1. Real Image and Matching Text:

$$\mathcal{L}_{\text{real}} = \mathbb{E}_{(x, e_{\text{sent}}) \sim P_r} [\max(0, 1 - D(x, e_{\text{sent}}))] \quad (3.3)$$

This term encourages the discriminator to output a high confidence score (at least 1.0) for real images that correspond to the correct textual descriptions.

2. Fake Image and Matching Text:

$$\mathcal{L}_{\text{fake}} = \mathbb{E}_{z \sim \mathcal{N}(0, I), (e_{\text{tok}}, e_{\text{sent}}) \sim P_r} [\max(0, 1 + D(G(z, e_{\text{tok}}, e_{\text{sent}}), e_{\text{sent}}))] \quad (3.4)$$

This term pushes the discriminator to assign low confidence score (at most -1.0) to generated images conditioned on the correct text. This encourages it to distinguish generated samples from real ones, even under matching conditions.

3. Real Image and Mismatched Text:

$$\mathcal{L}_{\text{mis}} = \mathbb{E}_{x \sim P_r, e_{\text{sent}} \sim P_{\text{mis}}} [\max(0, 1 + D(x, e_{\text{sent}}))] \quad (3.5)$$

This term pushes the discriminator to assign low confidence score (at most -1.0) to real images paired with incorrect (mismatched) text descriptions. This encourages it to detect semantic inconsistency and strengthens its understanding of text-image alignment.

4. Matching-Aware Gradient Penalty (MA-GP):

$$\mathcal{L}_{\text{MA-GP}} = \mathbb{E}_{(x, e_{\text{sent}}) \sim P_r} [|\nabla_{x, e_{\text{sent}}} D(x, e_{\text{sent}})|_2^6] \quad (3.6)$$

where the gradient with respect to both the image and text embedding is defined as:

$$\nabla_{x, e_{\text{sent}}} D(x, e_{\text{sent}}) = \left[\frac{\partial D(x, e_{\text{sent}})}{\partial x}, \frac{\partial D(x, e_{\text{sent}})}{\partial e_{\text{sent}}} \right] \quad (3.7)$$

This regularization term penalizes the sixth power of the ℓ_2 -norm of the discriminator’s gradients with respect to both the input image and its conditioning text embedding. It encourages the discriminator to have smooth gradients in the vicinity of real image-text pairs, which in turn helps the generator receive more stable and informative gradients as it learns to approach the real data manifold.

3.4.2 Generator Loss

The generator is trained to fool the discriminator by generating images that receive high scores when evaluated against their corresponding text descriptions. Formally, the generator loss is defined as:

$$\mathcal{L}_G = -\mathbb{E}_{z \sim \mathcal{N}(0, I), (e_{\text{tok}}, e_{\text{sent}}) \sim P_r} [D(G(z, e_{\text{tok}}, e_{\text{sent}}), e_{\text{sent}})] \quad (3.8)$$

By optimizing this loss, the generator learns to produce images that are both realistic and semantically aligned with the input text.

4 Results

In this section, we first introduce the training details and evaluation metrics used in our experiments, then evaluate the proposed model’s performance both quantitatively and qualitatively.

4.1 Training Details

Below, we detail the architectural and training choices that culminated in the final configuration used. Prior to identifying the best configuration, a total of 25 training runs were conducted with variations in resolution, learning rates, regularization strength, attention injection method and optimizer schedules. All subsequent quantitative and qualitative results refer to this final configuration reported in Table 4.1.

Model Settings. For the text encoder, we initially employed BERT-Mini as per assignment requirements; however, it exhibited limited alignment with fine-grained textual attributes. Consequently, after comparative experiments, the BERT-Mini encoder was replaced with the CLIP text encoder (ViT-B/32), which provided semantically richer embeddings and significantly improved visual-textual consistency.

The generator operates at a resolution of 128×128 . Initial experiments at 256×256 resulted in slower convergence and training instability; therefore, the lower resolution was adopted to enhance training stability and speed up experimentation. Upsampling is implemented via bilinear interpolation followed by a convolutional layer, which was preferred over transposed convolutions due to the latter’s tendency to produce checkerboard artifacts. Two conditioning strategies for cross-attention were evaluated: (i) direct fixed injection of conditioning information, and (ii) learnable conditioning strength parameters initialized to zero. Although the latter allows adaptive conditioning, it introduced instability and required longer training to converge. Hence, the fixed injection method was retained at the end.

Spectral normalization was applied to all convolutional layers in the discriminator, stabilizing training by enforcing Lipschitz continuity. Downsampling employed strided convolutions instead of pooling layers, as the former better preserve feature discriminability.

Optimization Settings. The network was optimized using Adam [9] with $\beta_1 = 0.0$ and $\beta_2 = 0.9$. The learning rate was set to 1×10^{-4} for the generator and 4×10^{-4} for the discriminator according to Two Timescale Update Rule (TTUR) [7]. The CLIP text encoder was fine-tuned with a smaller learning rate of 2×10^{-5} to preserve pretrained semantic representations. Various learning rate schedulers, including cosine annealing and linear decay, were explored, but fixed learning rates combined with manual early stopping consistently led to better results in practice.

The MA-GP regularization term is used to stabilize the discriminator’s behavior near real image-text pairs. After experimentation, the regularization strength was fixed to $\lambda_{\text{MA-GP}} = 1.5$, which balanced gradient smoothness and model expressiveness effectively.

The network was trained with a batch size of 64 using automatic mixed precision (AMP) to accelerate computation and reduce memory usage. AMP with `float16` precision was used for all adversarial objectives; however, numerically sensitive components such as the MA-GP term were computed in full precision (`float32`) to avoid instability.

Table 4.1: Hyperparameters configuration used to train the model.

Component	Value
Resolution	128×128
Batch Size	64
Text Encoder	CLIP (ViT-B/32)
Per-token Conditioning	Cross-attention (fixed)
Upsampling	Bilinear + Conv
Downsampling	Strided Conv
Discriminator Normalization	Spectral Norm
Optimizer	Adam ($\beta_1 = 0.0$, $\beta_2 = 0.9$)
Generator Learning Rate	1×10^{-4}
Discriminator Learning Rate	4×10^{-4}
Text Encoder Learning Rate	2×10^{-5}
$\lambda_{\text{MA-GP}}$	1.5

Training Monitoring. Training GANs presents unique challenges, particularly in maintaining equilibrium between the generator and the discriminator. If one network significantly overpowers the other, it can lead to failure modes such as *mode collapse* – where the generator produces a limited set of outputs regardless of input – or *training divergence* – where the gradients become unstable, causing the generator or discriminator loss to escalate uncontrollably and preventing convergence [4, 1]. These phenomena are common in GANs due to their adversarial dynamics, and as such, monitoring and diagnosing training stability is critical.

To this end, generator and discriminator losses were tracked throughout training using *TensorBoard*¹. Figure 4.1 presents the learning curves over time.

¹TensorBoard is a visualization toolkit for monitoring training metrics, part of the TensorFlow ecosystem: <https://www.tensorflow.org/tensorboard>

We can see that the as the discriminator gets better, the loss of the generator increases as a result, until they both stabilize and the entire GAN has practically converged after around 240 epochs (15000 steps). Both losses converge to a balanced plateau with bounded oscillations, an indicator of a successful adversarial equilibrium. Training was continued for approximately 200 additional epochs to ensure consistency without signs of collapse or divergence.

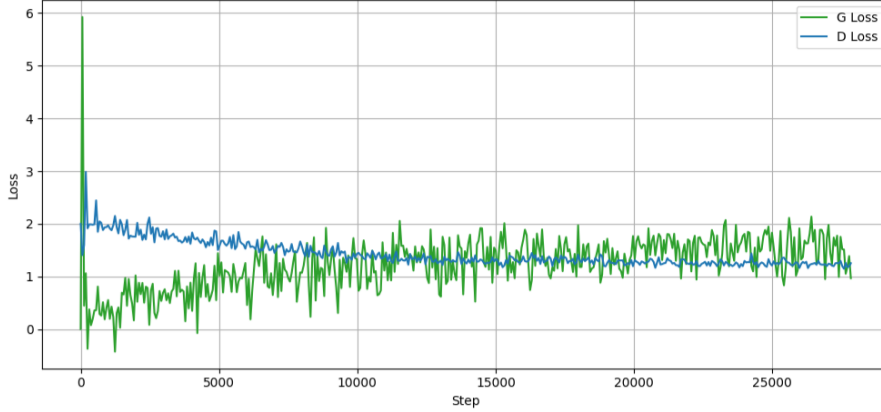


Figure 4.1: Training loss curves for the generator and discriminator.

Importantly, standard validation losses are not informative in GAN-based training, as they fail to reflect generation quality or mode coverage. Unlike traditional supervised settings, the GAN objective does not directly optimize for pixel-wise or likelihood-based error, rendering validation loss ineffective for early stopping or overfitting detection. Consequently, validation was conducted through a combination of qualitative and quantitative criteria. Visual inspection of generated samples over training epochs was periodically performed to identify signs of instability or degradation in sample quality. Quantitatively, the best checkpoint was selected based on the lowest FID computed on the held-out validation set. Both qualitative samples and FID values were logged and visualized using TensorBoard.

4.2 Evaluation Metrics

In this section, we describe in detail the metrics used to quantitatively assess fidelity, diversity and semantic alignment of generated images.

4.2.1 Inception Score (IS)

The *Inception Score* (IS) [18] was one of the first widely adopted quantitative metrics for evaluating generative models, particularly GANs. It seeks to quantify two desirable properties of generated images: individual sample quality and diversity across samples. Formally, it leverages a pretrained image classifier – typically

the *Inception v3* network trained on ImageNet [21] – to compute class probabilities for generated images, assuming these output predictions reflect meaningful semantic image content. Let $x \sim p_g(x)$ be an image generated from the model’s distribution. IS is computed by measuring the Kullback–Leibler (KL) divergence between the conditional class distribution $p(y|x)$ for a specific generated image (obtained from the pretrained classifier) and the marginal distribution $p(y) = \mathbb{E}_{x \sim p_g}[p(y|x)]$ averaged over all generated samples. The score is then exponentiated:

$$\text{IS}(G) = \exp \left(\mathbb{E}_{x \sim p_g} [\text{D}_{\text{KL}}(p(y|x) \parallel p(y))] \right)$$

The KL divergence quantifies how much the prediction for an individual image deviates from the average prediction across the entire batch of generated images: if $p(y|x)$ has low entropy (i.e., the classifier makes confident predictions) and if $p(y)$ has high entropy (i.e., predictions are spread across many classes), then the KL divergence will be large and thus the IS will be high. Therefore, this aligns with the two above desired properties.

To compute IS, in practice, thousands of generated samples are passed through the pretrained classifier and empirical estimates of $p(y|x)$ and $p(y)$ are used to compute the score. A higher IS is generally considered better.

While IS is intuitive and easy to compute, it suffers from several limitations:

- It does not directly compare the generated distribution to the real data distribution. A model could overfit or collapse to a few high-confidence samples and still get a good IS.
- It relies on the alignment between the pretrained classifier’s label space (e.g., ImageNet classes) and the data being generated, which limits its applicability to non-natural image domains.
- It has been shown that IS can be artificially inflated without genuinely improving sample quality.

Because of these shortcomings, IS has largely been replaced in modern GAN evaluation by more robust alternatives such as FID.

4.2.2 Fréchet Inception Distance (FID)

The **Fréchet Inception Distance (FID)** [7] has become a standard metric for evaluating the fidelity and diversity of images produced by generative models. In contrast to IS, which evaluates generated images independently of real ones, FID directly compares the distributional statistics of generated samples to those of real ones.

To compute FID, both real and generated images are passed through a pre-trained Inception v3 network, and activations are extracted from a designated intermediate layer, commonly the final average pooling layer, which produces 2048-dimensional feature vectors. These features for each domain are then approximated as samples from multivariate Gaussian distributions. Let μ_r, Σ_r denote the empirical mean and covariance of the features extracted from real images, and μ_g, Σ_g those of the generated images. The FID is defined as the Fréchet distance (also known as the Wasserstein-2 distance) between these two Gaussian distributions:

$$\text{FID}(X_r, X_g) = \|\mu_r - \mu_g\|_2^2 + \text{Tr} \left(\Sigma_r + \Sigma_g - 2(\Sigma_r \Sigma_g)^{1/2} \right)$$

Here, the first term measures the distance in mean features, capturing differences in the average features (content similarity), while the second term measures the distance related to the covariance matrices, reflecting differences in how features vary and correlate (diversity similarity).

In opposite to IS, a lower FID indicates that the generated samples closely resemble real ones in both feature content and distributional structure.

FID addresses several of the key limitations of the Inception Score (IS):

- It directly compares the distribution of generated data to that of real data.
- It is more sensitive to mode collapse, effectively detecting situations where the model produces insufficiently diverse outputs.
- It is generally more robust to noise and visual artifacts, since it evaluates similarity in a learned feature space rather than relying on classification confidence alone.

However, FID is not without its limitations:

- It relies on the Inception v3 feature space, which may not align well with the semantics of non-ImageNet datasets.
- It assumes that image features follow a multivariate Gaussian distribution in the feature space, an approximation that may not strictly hold in practice.
- The computed score is sensitive to the number of samples used; meaningful comparisons require using the same sample size – typically 10,000 or 50,000 images – for both real and generated sets.
- With small sample sizes, the empirical estimation of means and covariances can become unreliable, introducing bias into the FID measurement.

4.2.3 Kernel Inception Distance (KID)

The *Kernel Inception Distance* (KID) [2] is a metric for evaluating generative models that addresses key limitations of FID. FID compares the statistics of real and

generated image features using a multivariate Gaussian approximation. It assumes that these features are normally distributed and can yield biased estimates when computed on small sample sizes.

KID mitigates these issues by employing the *Maximum Mean Discrepancy* (MMD), a non-parametric kernel-based measure that compares the full distributions of feature embeddings without assuming any specific parametric form. Like FID, it relies on features extracted from an intermediate layer of a pretrained Inception v3 network (typically the final pooling layer, producing 2048-dimensional vectors), but it quantifies the difference between real and generated distributions using kernel methods.

Formally, given real samples $\{x_i\}_{i=1}^m \sim P_r$ and generated samples $\{y_j\}_{j=1}^n \sim P_g$, KID is defined as the squared MMD between the two distributions using a kernel function k :

$$\text{KID}(X, Y) = \text{MMD}^2(X, Y) = \mathbb{E}_{x, x'}[k(x, x')] + \mathbb{E}_{y, y'}[k(y, y')] - 2\mathbb{E}_{x, y}[k(x, y)],$$

where $x, x' \sim P_r$, $y, y' \sim P_g$ and the kernel k is typically chosen as a third-degree polynomial:

$$k(x, y) = \left(\frac{1}{d} x^\top y + 1 \right)^3,$$

with d being the dimensionality of the feature vectors.

In practice, the KID is estimated by computing the MMD over multiple random subsets (splits) of the real and generated datasets. The final score is reported as the mean and standard deviation averaged across these splits, providing a robust and unbiased estimate of distributional similarity.

As with FID, lower KID values indicate that the distribution of generated features is closer to that of the real ones, implying better visual quality and semantic diversity. Unlike FID, KID remains statistically unbiased even for small sample sizes and does not rely on Gaussian assumptions, making it a more reliable metric in data-limited settings.

4.2.4 CLIP Score

The *CLIP Score* [6] is a more recent metric designed to evaluate the semantic alignment between generated images and their corresponding text descriptions. CLIP Score leverages the CLIP model, which jointly embeds images and text into a shared feature space.

Let x denote a generated image and t the corresponding textual description. The CLIP model provides normalized embeddings e_x and e_t for the image and text, respectively. The CLIP Score is computed as the cosine similarity between these embeddings:

$$\text{CLIPScore}(x, t) = \max(100 \cdot \cos(e_x, e_t), 0) \quad (4.1)$$

where

$$\cos(e_x, e_t) = \frac{\langle e_x, e_t \rangle}{\|e_x\|_2 \|e_t\|_2} \quad (4.2)$$

The resulting score is bounded between 0 and 100, where higher values indicate stronger semantic alignment between the image and the text.

Empirical studies have shown that CLIP Score is well correlated with human judgment. However, it evaluates only the alignment between image and text, and does not directly account for image realism, fidelity, or diversity. Therefore, it is typically used alongside complementary metrics such as FID or KID to obtain a more comprehensive assessment of generative model performance.

4.3 Quantitative and Qualitative Evaluation

As anticipated in Section 2.5, we evaluate the generative performance of our model using two distinct test sets:

- **Intra-Test Set:** Designed to assess the model’s robustness to lexical and syntactic variation. It contains test entries with paraphrased descriptions of training samples. These paraphrases were never seen during training and are used to evaluate the model’s ability to generate accurate images when given semantically equivalent, but syntactically different text inputs.
- **Novel-Test Set:** Intended to measure the model’s generalization capability. It includes entirely novel textual descriptions corresponding to unseen visual concepts – that is, none of these image-text pairs were present during training. This setup evaluates the model’s capability to synthesize plausible outputs based solely on its learned textual and visual priors.

On both test sets, we conduct a comprehensive evaluation using a combination of *quantitative metrics* – including KID, IS, FID and CLIP Score – as well as *qualitative analysis*, which involves human visual inspection and interpretation of the generated outputs. This dual approach allows us to assess both the statistical fidelity and the perceptual realism of the generated images under varying degrees of input novelty.

4.3.1 Quantitative Results

Given the modest size of the test sets (403 samples for Intra-Test and 402 for Novel-Test), we consider KID to be the most reliable metric to consider as it provides an unbiased estimate even for smaller sample sizes. For our experiments, KID was computed using 100 random subsets to ensure a stable and robust estimate.

Nonetheless, we also report the IS and FID for completeness, although these may be less reliable under such constrained evaluation conditions. To assess semantic alignment between generated images and their corresponding textual descriptions, we adopt the CLIP Score as our sole text-image alignment metric.

KID, IS and FID metrics were calculated using the *torch-fidelity*² library, while CLIP Score was computed using the *torchmetrics*³ library.

Table 4.2 summarizes the results across all four metrics for both test sets.

Table 4.2: Quantitative results on Intra- and Novel-Test sets.

Metric	Intra-Test	Novel-Test
KID (\downarrow)	0.047 ± 0.00001	0.068 ± 0.00001
Inception Score (\uparrow)	2.936 ± 0.321	2.642 ± 0.244
FID (\downarrow)	107.404	117.837
CLIP Score (\uparrow)	25.416	24.580

The Intra-Test results suggest that the model is capable of generating moderately diverse and visually plausible images when conditioned on unseen paraphrased descriptions of concepts encountered during training. However, overall performance remains limited, reflecting the challenges of achieving strong generalization with limited data.

As expected, performance further deteriorates in the Novel-Test setting. All three distributional metrics — KID, IS and FID — worsen compared to the Intra-Test, highlighting the increased challenge of synthesizing realistic images from entirely novel text descriptions.

The CLIP Score remains relatively stable across the two test sets. However, its overall magnitude indicates only poor-to-moderate semantic alignment, reflecting the model’s limited capacity to faithfully capture fine-grained textual details under the current training regime.

4.3.2 Qualitative Evaluation

We now turn to a qualitative evaluation of generated images under both the Intra-Test and Novel-Test settings. Tables 4.3 and 4.4 show a selection of generated vs. original images for unseen paraphrased descriptions of seen concepts (Intra-Test), while Table 4.5 presents results for entirely unseen descriptions (Novel-Test).

Intra-Test Observations. Overall, the model often succeeds at capturing the most salient attributes of familiar concepts, especially overall shape and dominant

²<https://github.com/toshas/torch-fidelity>

³<https://github.com/Lightning-AI/torchmetrics>

color. For instance, it effectively renders Charizard’s orange body and wing structure, as well as Scyther’s green insectoid form and blade-like arms, making them broadly recognizable. However, these samples are *cherry-picked* to showcase the model’s best outputs. In many other Intra-Test cases (not shown), the network struggles to reproduce crucial details, leading to *distortions*, *missing limbs* or even *incorrect color assignments*. This indicates that even slight variations or paraphrasing of familiar descriptions can introduce fragility into the model’s generative performance.

Table 4.3: Intra-Test Qualitative Evaluation (1/2). Each row presents a Pokémon with its original textual description, a paraphrased version seen during training and a novel paraphrase used for testing. The corresponding generated image (from the test paraphrase) is shown alongside the original reference image.

















Name	Original Description	Train Paraphrase	Test Paraphrase	Generated Image	Original Image
Charizard	A towering, dragon-like creature with an orange body, cream belly, and powerful wings with blue-green undersides. Its long neck and horned head add to its fearsome appearance. A blazing tail and sharp claws complete its intimidating, sky-dwelling form.	This creature is a formidable, dragon-like figure with imposing proportions. Its orange body, cream belly, and wingspan are impressively long and blue-green. Additionally, it has horned head and long neck, as well as blazing tail and claws that add to its intimidating appearance.	It is a formidable, dragon-like creature with an orange body, curved shoulders, cream skin, and wings that have blue-green undersides. Its long neck and horned head are both impressive. A tail that is hot as lightning and its sharp claws complete its intimidating appearance.		
Scyther	A large, green insectoid with scythe-shaped forearms, large wings, and sharp mandibles. Its sleek body and agile form enable swift, slicing movements.	This insectoid is a large green insect with broad wings, scythe-shaped forearms, and sharp mandibles. Its agile posture and smooth body make it easy to slice through its body quickly.	A slender green insectoid with broad wings, forearms that resemble a hawk, and sharp mandibles. Its agile posture and smooth muscles allow for quick, blade-crushing actions.		
Psyduck	A duck-like creature standing on two legs, with yellow fur, a round body, and a wide cream beak. Its blank stare, tiny pupils, and three black hairs atop its head give it a perpetually dazed and confused look.	This is a duck-like creature standing on two legs, with yellow fur covering its entire body and curved cream beak. Its uncomplicated expression, minute pupils, and three black hairs on its head make it appear constantly disoriented and dazed.	A duck-like creature with yellow fur, a round body and wide cream beak. Its eyes are blank, its pupils are small, and its head is covered in black hair. The creature stands on two legs and has an uncontrollable countenance.		
Electrode	A smooth, spherical form resembling an inverted Poké Ball with a red bottom half and white top. It has a wide mischievous grin and a glossy, polished surface that gleams brightly.	A slick, round shape resembling an inverted Poké Ball with a red bottom half and white top. It has pronounced cheekbones and gleaming eyes and shiny, polished surfaces.	It is a smooth, round shape that looks like an inverted Poké Ball with 'the bottom half red and the top white' A wide grin: All over shiny and polished.		













Table 4.4: Intra-Test Qualitative Evaluation (2/2). Each row presents a Pokémon with its original textual description, a paraphrased version seen during training and a novel paraphrase used for testing. The corresponding generated image (from the test paraphrase) is shown alongside the original reference image.

Name	Original Description	Train Paraphrase	Test Paraphrase	Generated Image	Original Image
Gengar	A round, purple figure with a wide, wicked grin and sharp pointed ears. Its bright red eyes gleam with mischief, and spiky protrusions line its back, enhancing its eerie, playful aura.	A purple-colored object with a wide, wicked smile and pointed, sinister ears. Its brilliant red eyes are filled with mischief, and its back is lined with sharp, jagged protrusions.	An enigmatic, purple creature with a wide, wicked grin and pointed ears. Its bright red eyes are filled with mischief, and its back is lined with sharp indentations of spiky hair.		
Oddish	A tiny, round creature with deep blue skin, glowing red eyes, and a small smiling mouth. From the top of its head sprout five wide, leafy green blades, giving it the appearance of a walking plant bulb or animated weed emerging from the earth.	A small, round animal with deep blue skin, luminous red eyes, and a tiny smile. Its head is covered in five broad green blades, giving it the appearance of shedding weed or bulbous plant bulbs.	This is a small, circular animal with deep blue skin, red gleaming eyes and minuscule smile. Five broad, leafy green blades grow from the tip of its head, giving it an appearance of pulverized weed or ambulatory plant bulbs.		
Pikachu	A small, round-cheeked rodent with bright yellow fur and two brown stripes on its back. Its long ears have black tips, and its face features large eyes and red circular cheeks. A jagged, lightning-shaped tail with a brown base gives it a lively, energetic silhouette.	This is a small, round-cheeked, bright yellowish-fur rodent with two brown stripes on its back with long, black tips of fur. Its face has large eyes and red circular cheeks and twirling eyes, while its jagged, lightning-shaped tail with striped legs creates an energetic, look that can be worn all day.	With a bright yellow fur and two brown stripes on its back, this small rodent has long ears with black tips. Its face is distinguished by large eyes and red circles on the cheeks. A jagged, lightning-shaped tail with contrasting brown base gives it vigor.		
Squirtle	A small turtle-like creature with light-blue skin, big purplish eyes, and a curled tail. Its sturdy brown shell has a pale yellow underside and thick white edging, giving it a tough, playful look.	A tiny turtle-like creature with a light-blue complexion, sizable purplish eyes, and curved tail. Its sturdy brown shell has accumulated globules on its underside, which are now covered in contrasting white paint, creating an aggressive and playful appearance.	The small, turtle-like creature has large purplish eyes and a curled tail. Its sturdy brown shell has pheromone patches on its underside and thick white bordering, giving it ferocity.		

Novel-Test Observations. In the Novel-Test setting – where the model generates images from descriptions of concepts unseen during training – the outputs naturally deviate more significantly from the originals. Despite this, the model often retains key color schemes, such as the pink of Cleffa or the blue and yellow of Croconaw, and occasionally captures secondary features like the ice-shard wings of Kyurem. These successes suggest the generator has learned general associations

between color terms and certain distinctive geometric patterns. Nonetheless, many fine details are lost or overly simplified, often resulting in a blurred or less defined image.

Table 4.5: Novel-Test Qualitative Evaluation. Each row shows a Pokémon whose description was never seen during training. The model generates an image based solely on the provided text description, which is compared with the original reference image.

Name	Description	Generated Image	Original Image
Buneary	Small and rabbit-like with curled brown ears. Its upper body has smooth chocolate-colored fur, while the lower half is covered in light tan fleece.		
Cacturne	A humanoid cactus resembling a scarecrow with spiky limbs, green rhombi patterns, a grinning face, and a spiked triangular hat-like head.		
Cleffa	A small, pink creature with a star-like shape and curled tail. It has stubby limbs, a cheerful face, and a swirl on its forehead, giving it a charming and innocent appearance.		
Croconaw	A stout, blue reptilian creature with a yellow jaw and red spikes on its head, back, and tail. Its body has an uneven blue-and-yellow pattern, and its large jaws are filled with slanted fangs.		
Darmanitan	A squat, ape-like figure with a red body and tan face. It has bushy flame-shaped eyebrows, spiked teeth, and strong limbs for an energetic appearance.		
Kyurem	A towering gray dragon with icy features, jagged wings, yellow glowing eyes, and shards of ice along its body.		

Common Failure Modes. Across both test sets, we observe that:

- The model preferentially encodes **color** and **basic shape** over less salient attributes.

- **Spatial relationships** (exact placement of limbs, horns, tails) can be inconsistent, leading to visual artifacts.
- **Fine textures** and **decorative elements** (spikes, swirls, patterns) are often omitted or rendered inaccurately.
- In extreme cases, the network produces **garbled shapes** that bear little resemblance to the intended creature.

These qualitative patterns help explain the moderate CLIP Scores (≈ 25) observed: while prominent attributes are often present, the generator fails to represent the **full spectrum of textual details** needed for strong semantic alignment. Likewise, the diversity of quality – from plausible to unintelligible – contributes to the relatively high FID and KID values reported earlier.

5 Conclusion

This project focused on the development of a text-to-image synthesis model specifically tailored to generate Pokémon-style sprite images from their respective Pokédex descriptions. This was addressed through two main contributions: (1) the creation of a custom Pokémon dataset to mitigate the limitations of the original data and (2) the design and implementation of a conditional GAN leveraging both global sentence and token-level textual conditioning from a fine-tuned text encoder.

The resulting model demonstrated promising capabilities in translating textual descriptions into coherent visual outputs. It consistently captures dominant visual attributes such as color schemes and coarse shapes, yet exhibits limited performance in reproducing fine-grained details and spatial precision, which hinder the generation of highly accurate and semantically rich images.

Currently, token-level textual conditioning is applied exclusively within the generator, while the discriminator operates solely on the global sentence embedding. This design choice was motivated by computational efficiency, as extending token-level conditioning to the discriminator would substantially increase training time. Nevertheless, extending token-level conditioning to the discriminator represents a logical avenue for future work, potentially enhancing semantic alignment discrimination capabilities.

Additionally, the integration of self-attention mechanisms within both generator and discriminator architectures could improve spatial coherence and fine detail generation. Such mechanisms were not included in the present work due to their well-known quadratic computational complexity, which poses practical challenges given limited resources.

The project faced several challenges, foremost among them the limited size of the dataset. Expanding the dataset remains a critical step to improve model generalization and output diversity. Moreover, exploring adaptive augmentation strategies for GANs such as *Differentiable Augmentation* (DiffAugment) [25] or *Adaptive Discriminator Augmentation* (ADA) [8] could enhance training robustness. These techniques are specifically designed to prevent discriminator overfitting in low-data regimes by applying augmentations – differentiable in DiffAugment, and adaptively controlled in ADA – directly to the discriminator’s inputs, thereby improving generalization without requiring additional real data. However, these techniques require careful consideration to avoid introducing semantic misalignment between augmented images and their corresponding textual descriptions.

Another aspect warranting further investigation is the nature of the textual descriptions and their paraphrases. Simplifying or shortening these inputs may help reduce semantic ambiguity and improve alignment between text and generated images.

Training dynamics revealed that the discriminator struggled most with mismatch loss, indicating difficulty in correctly associating images with their corresponding textual descriptions. Addressing this issue through improved loss formulations or enhanced conditioning mechanisms could yield better semantic consistency.

Future work could also explore incorporating auxiliary loss functions, such as CLIP loss, to strengthen semantic alignment, or perceptual loss to improve image quality and fidelity.

Lastly, comprehensive hyperparameter optimization, potentially via Bayesian optimization frameworks, could systematically identify more effective training configurations, leveraging partial training runs to reduce computational cost.

In conclusion, while this project establishes a foundation for Pokémon sprite generation conditioned on textual descriptions, there remain multiple avenues for refinement and extension to achieve more precise and semantically faithful image synthesis.

Bibliography

- [1] Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein gan, 2017.
- [2] Mikołaj Bińkowski, Danica J Sutherland, Michael Arbel, and Arthur Gretton. Demystifying mmd gans. *arXiv preprint arXiv:1801.01401*, 2018.
- [3] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 conference of the North American chapter of the association for computational linguistics: human language technologies, volume 1 (long and short papers)*, pages 4171–4186, 2019.
- [4] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014.
- [5] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.
- [6] Jack Hessel, Ari Holtzman, Maxwell Forbes, Ronan Le Bras, and Yejin Choi. Clipscore: A reference-free evaluation metric for image captioning. *arXiv preprint arXiv:2104.08718*, 2021.
- [7] Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. Gans trained by a two time-scale update rule converge to a local nash equilibrium. *Advances in neural information processing systems*, 30, 2017.
- [8] Tero Karras, Miika Aittala, Janne Hellsten, Samuli Laine, Jaakko Lehtinen, and Timo Aila. Training generative adversarial networks with limited data. *Advances in neural information processing systems*, 33:12104–12114, 2020.
- [9] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.
- [10] Alexander Kirillov, Eric Mintun, Nikhila Ravi, Hanzi Mao, Chloe Rolland, Laura Gustafson, Tete Xiao, Spencer Whitehead, Alexander C Berg, Wan-Yen Lo, et al. Segment anything. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 4015–4026, 2023.
- [11] Jae Hyun Lim and Jong Chul Ye. Geometric gan. *arXiv preprint arXiv:1705.02894*, 2017.
- [12] Shilong Liu, Zhaoyang Zeng, Tianhe Ren, Feng Li, Hao Zhang, Jie Yang, Qing Jiang, Chunyuan Li, Jianwei Yang, Hang Su, et al. Grounding dino: Marrying dino with grounded pre-training for open-set object detection. In *European conference on computer vision*, pages 38–55. Springer, 2024.
- [13] Mehdi Mirza and Simon Osindero. Conditional generative adversarial nets. *arXiv preprint arXiv:1411.1784*, 2014.
- [14] Takeru Miyato, Toshiki Kataoka, Masanori Koyama, and Yuichi Yoshida.

- Spectral normalization for generative adversarial networks. *arXiv preprint arXiv:1802.05957*, 2018.
- [15] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. Learning transferable visual models from natural language supervision. In *International conference on machine learning*, pages 8748–8763. PmLR, 2021.
- [16] Tianhe Ren, Shilong Liu, Ailing Zeng, Jing Lin, Kunchang Li, He Cao, Jiayu Chen, Xinyu Huang, Yukang Chen, Feng Yan, et al. Grounded sam: Assembling open-world models for diverse visual tasks. arxiv 2024. *arXiv preprint arXiv:2401.14159*.
- [17] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 10684–10695, 2022.
- [18] Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. Improved techniques for training gans. *Advances in neural information processing systems*, 29, 2016.
- [19] Mike Schuster and Kuldeep K Paliwal. Bidirectional recurrent neural networks. *IEEE Transactions on Signal Processing*, 45(11):2673–2681, 1997.
- [20] Rico Sennrich, Barry Haddow, and Alexandra Birch. Neural machine translation of rare words with subword units. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1715–1725, 2016.
- [21] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2818–2826, 2016.
- [22] Ming Tao, Hao Tang, Fei Wu, Xiao-Yuan Jing, Bing-Kun Bao, and Changsheng Xu. Df-gan: A simple and effective baseline for text-to-image synthesis. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 16515–16525, 2022.
- [23] Yonghui Wu, Mike Schuster, Zhifeng Chen, et al. Google’s neural machine translation system: Bridging the gap between human and machine translation. In *arXiv preprint arXiv:1609.08144*, 2016.
- [24] Tao Xu, Pengchuan Zhang, Qiuyuan Huang, Han Zhang, Zhe Gan, Xiaolei Huang, and Xiaodong He. Attngan: Fine-grained text to image generation with attentional generative adversarial networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1316–1324, 2018.
- [25] Shengyu Zhao, Zhijian Liu, Ji Lin, Jun-Yan Zhu, and Song Han. Differentiable augmentation for data-efficient gan training. *Advances in neural information processing systems*, 33:7559–7570, 2020.