Mid-Semester Project Report

Literature Review

Paper Chosen

DreamBooth: Fine Tuning Text-to-Image Diffusion Models for Subject-Driven Generation

Summary of Contributions

DreamBooth addresses a core limitation of modern text-to-image diffusion models. (e.g., DALL·E 2, Imagen, Stable Diffusion) That is their inability to consistently generate a specific real-world subject—such as a particular dog, person, or object—across different scenes and contexts.

The main contribution of DreamBooth is a fine-tuning technique that allows users to “implant” a unique subject into a pre-trained diffusion model. With just a few (3–5) reference images, the model learns to generate new, diverse images of the subject while preserving its identity and features. This enables personalization in image generation using minimal data.

Key Ideas

1. **Personalization of Large Text-to-Image Models**
   DreamBooth extends a pre-trained diffusion model so that it can produce images of a *new subject* (for instance, “my special mug” or “my cat Bruno”), using only a handful (3–5) of reference images.

2. **Unique Identifier**
   The authors introduce a *unique token*, usually a made-up word, that stands for the subject. For example, instead of prompting “a dog,” users might prompt “a [V] dog,” where “[V]” is a newly learned token that specifically represents the user’s dog. That

token is “rare” or “nonsense” in the language space, so the diffusion model is less likely to confuse it with existing words.

3. **Few-Shot Fine-Tuning**
   DreamBooth fine-tunes *all* of the model’s parameters with a small set of reference images labeled with the new token. This allows it to preserve the distinctive features of the subject. The fine-tuning requires surprisingly few iterations—often on the order of hundreds to a thousand steps.

4. **Class-Specific Prior Preservation Loss**
   To prevent the model from forgetting the general category (e.g., “dogs” in general) during fine-tuning, DreamBooth introduces a class-specific prior preservation loss. This maintains diversity and avoids overfitting to the reference subject.

Methods and Findings

- **Few Reference Images**:
  Each subject typically requires only 3–5 images. These images might show different angles or settings, but it is not necessary to have an extensive dataset.

- **Fine-Tuning Setup**:

    - A *rare token* is automatically chosen from the underlying text tokenizer so that it has minimal prior meaning in the model.

    - Each of the subject’s images is captioned “A [V] [class noun].” For example, “A [V] backpack” if the subject is a backpack.

    - Parallel to learning from the subject images, the model also generates “fake” examples of the broader class (“A backpack”) using the *original* pre-trained model; these generated examples are then used in a **prior-preservation loss** to make sure the model does not collapse or overfit.

- **Qualitative Results**:
    - Once trained, the prompt “A [V] backpack in the Sahara desert” yields a new image in which the backpack is realistically placed in a desert scene—maintaining its color, shape, and distinctive features.
    - The authors demonstrate a range of use cases: having a particular dog appear in various art styles, merging features of a user’s pet with a different animal species, or generating brand-new “photographs” of a single real person in multiple settings.
- **Quantitative Evaluations**:
    - **Subject fidelity** is tested by measuring how similar the generated images are to the original subject images, in terms of distinctive details (e.g., the shape of a teapot or stripes on a cat).
    - **Prompt fidelity** is tested by measuring how well the generated images reflect the text prompt.

Critical Assessment

Strengths

- **Data Efficiency**: Achieves high-quality personalization with only a few images.
- **Versatility**: Works with various subjects—objects, pets, and people alike.
- **Visual Diversity**: Maintains the ability to generate varied images thanks to the prior-preservation loss.
- **Usability**: Once fine-tuned, the model can synthesize high-quality outputs for a range of prompts involving the subject.

Limitations

- **Overfitting**: If fine-tuning runs too long or the prompts closely resemble the original training images, the generated outputs can become overly similar to the references.

- **Entangling Context**: When the user's reference photos are visually similar (e.g., the subject always in the same room), the model may entangle background details with the subject.

- **Complex or Rare Classes**: For unusual objects, or if the class is not well-represented in the original diffusion model's training data, DreamBooth's effectiveness can drop.

- **Compute and Resource Costs**: Although DreamBooth typically takes only minutes to fine-tune on a GPU/TPU, it still requires enough resources to run a training loop—rather than using a purely "prompt-only" method.

Baseline Design

Data Preprocessing

DreamBooth requires relatively little data — typically only 3 to 5 images of a target entity (such as a specific dog or person) are sufficient for personalized training. The preprocessing involves the following steps:

1. Instance Images

Instance images are the specific photos of the subject you want the model to learn and generate. Here, these images are the main data used to teach the model a new concept — such as a particular person, pet, object, or style.

All images must be resized to a uniform size (512×512) to match the input requirements of Stable Diffusion. Below are the instance images I choose for the model to learn.

2. Class Prior Preservation (Regularization Images)

Regularization Images are **generic images of a class** (like “dogs”, “cats”, “people”) that are not of my specific instance, and they are used to help the model **retain its general knowledge** of that class during fine-tuning. When we train DreamBooth on just 3–5 instance photos, the model may **overfit** — meaning it might "forget" what the general class (like "cat") looks like, and only generate your specific cat every time you prompt for "a cat"(losing the ability to generalize).

As a result, we also need to prepare photos which are in the same class of our instances, still, they need to be resized to 512×512. Below are the regularization images.

3. Prompt Engineering

Each image needs to be paired with a descriptive text prompt.

The prompt should include a unique identifier token, in our example, we use **"aaa dog"**, where **"aaa"** is a custom token and **"cat"** indicates the class.

For example: we used prompt "a aaa cat sitting in Times Square, New York City, with buildings and billboards around." Below is the result from the model.

Model Architecture

DreamBooth is a **fine-tuning method** built on top of an existing large-scale text-to-image diffusion model. Here's how we choose and use the baseline model:

1. Pretrained Model Architecture

In the paper, the author used a pretrained text-to-image which is not open source. In our work, we chose the state-of-the-art **Stable Diffusion Model**(stable diffusion v1.5), which is a pretrained latent text-to-image diffusion model and can be downloaded from https://huggingface.co/stable-diffusion-v1-5/stable-diffusion-v1-5.

The stable diffusion model consists 3 main parts, Text Encoder, U-Net and VAE, here is the introduction to all of them.

- Text Encoder converts text prompt into vector embeddings using **CLIP (ViT-L/14).** These embeddings condition the image generation process and the encoder is frozen during training and fine-tuning.
- U-Net is the core denoising network that predicts and removes noise from latent features. It is the heart of the model and operates in the **latent space**, which is combined with text embedding and optional timestep embedding. Its input is Noisy latent image + CLIP text embedding and output is predicted noise to subtract at each diffusion timestep. It follows the classical U-Net architecture:

Part	Description
Downsampling Path	Extracts hierarchical features via ResBlocks and attention layers
Bottleneck	Middle block with self-attention

Upsampling Path	Reconstructs features, with skip connections from encoder path
Attention Layers	Applied at multiple resolutions to inject text conditioning

- VAE (Variational Autoencoder) converts images to/from latent space (used for compression and reconstruction). The VAE **encodes** 512×512 pixel images into a latent space of size 64×64×4 and this latent space is where the diffusion process operates. After generation, the VAE **decodes** latent images back to pixel space.

2. Fine-Tuning Architecture

We took the similar strategy that Dreambooth use for fine-tuning, which can be summarized as follows: 1. Insert a new token into the tokenizer 2. Initialize its embedding randomly
3. Pair images with prompts like “a [aaa] cat” 4. Fine-tune the token embedding and selected parts of the U-Net 5 .Use instance loss + class prior loss to learn the new subject and avoid forgetting general category knowledge.
The fine-tuning architecture we chose can be called Full U-Net + Token Embedding Fine-Tuning. Here is the summarization of its behaviors.

Module	Trained?	trainable parameters
U-Net (Denoiser)	Yes	All U-Net weights — convolution layers, attention layers, normalization layers
Token Embedding	Yes	Embedding vector(s) for the custom token
CLIP Text Encoder	No	None
VAE (Autoencoder)	No	None
Class Prior Preservation	Yes	Affects instance vs class image loss balancing (but not model weights)

Hyperparameters

Hyperparameter	Value	Description
--project_name	Proj_cats	Name of the training project; used for organizing outputs and checkpoints
--training_model	v1-5-pruned-emaonly-pruned.ckpt	Pretrained Stable Diffusion v1.5 checkpoint (base model)
--training_images	Proj/images/samples/samples_cats	Directory containing (specific cat) images
--regularization_images	Proj/images/regularization/reuglarized_cats	Generic class images of the same category (e.g. cats) used for class prior preservation
--class_word	“Cat”	General class name used in regularization prompts
--token	“aaa”	Custom identifier token that the model learns to associate with the specific cat
--flip_p	0.0	No image flipping during training
--learning_rate	1e-6	A very conservative learning rate, good for slow, stable convergence
--max_training_steps	2000	Number of training steps; sufficient for 3–5 images
--save_every_x_steps	500	Saves intermediate checkpoints at every 500 steps
--optimizer	AdamW	Adaptive optimizer with weight decay
--EMA (Exponential Moving Avg)	False	EMA is disabled, so model updates are direct
--Regularization Loss	1.0	Class loss weight (instance loss : class

Weight		loss = 1:1)
--unfreeze_model	True	The entire U-Net is trainable (full fine-tuning)
--cond_stage_trainable	True	Required for DreamBooth to support token embedding learning
--embedding_reg_weight	0.0	Not using extra L2 regularization on the token embeddings
--per_image_tokens	False	Uses one shared token embedding for all instance images

Evaluation Metrics

We evaluate the quality of the generated images using two CLIP-based metrics: **text-to-image similarity** and **image-to-image similarity**. The former measures how well the generated images align with the input prompt, while the latter evaluates how visually similar the outputs are to the instance images. Both metrics are computed using the CLIP ViT-B/32 model by comparing the cosine similarity between normalized embeddings.

1. text-to-image similarity

Goal: Measure how well the generated images match the intended textual prompt (e.g., “a aaa cat sitting in Times Square, New York City, with buildings and billboards around”).
Algorithm:

- **Text Tokenization**
  The input prompt string is tokenized using CLIP's tokenizer and converted into embeddings using the text encoder.
- **Image Preprocessing**
  Each generated image (normalized to [–1, 1]) is preprocessed to match CLIP's expected input format:
  a. Rescaled to [0, 1]
  b. Resized and center-cropped (typically to 224×224)

c. Normalized using CLIP-specific mean and std

- **Feature Encoding**
  a. The prompt is encoded into a **text embedding** vector $T \in \mathbb{R}^d$
  b. The generated images are encoded into **image embeddings** $I_1, I_2, \ldots, I \in \mathbb{R}^d$
- **Normailzation**
  All embeddings are L2-normalized to ensure cosine similarity equals dot product.
- **Similarity Computation**
  For each image embedding I_i, compute cosine similarity with T: $sim_i = (T \cdot I_i) / (||T|| * ||I_i||) \approx T \cdot I_i$

Interpretation: a. Higher values (closer to 1.0) indicate stronger alignment with the text prompt. b. Useful for evaluating prompt conditioning quality.

2. image-to-image similarity

Goal: Measure how visually similar the generated images are to the original instance images (i.e., preservation of identity or concept).

Algorithm:

- **Input Preparation**
  Both the **reference instance images** and the **generated images** are preprocessed in the same way as above.
- **Feature Encoding**
  a. Each reference image is encoded into a CLIP image embedding $R_1, R_2, \ldots, R \in \mathbb{R}^d$
  b. Each generated image is encoded into $G_1, G_2, \ldots, G \in \mathbb{R}^d$
- **Normalization**
  All image embeddings are L2-normalized.
- **Similarity Computation**
  Compute the mean cosine similarity between the two sets: Img2Img_Similarity = (1/mn) ∑ ($R_i \cdot G$□) for all i in [1, m], j in [1, n]

Interpretation: a. Higher scores indicate that the generated images are visually close to the training instance images. b. Especially useful for evaluating identity preservation in personalized generation.

Preliminary Results

Comparision Between Different Prompts

Here are the three prompts and their results.

1. Prompt: a aaa cat at Duke University, Durham

2. Prompt: a aaa cat at New York

3. Prompt: a aaa cat sitting in Times Square, New York City, with buildings and billboards around

Observation:

For first two prompts: Prompt 1: *"a aaa cat at Duke University, Durham"*
Prompt 2: *"a aaa cat at New York"*
→ The generated images look very similar — same background, same pose, limited variation.

But for: Prompt 3: *"a aaa cat sitting in Times Square, New York City, with buildings and billboards around"*
→ The images are much more diverse, with bright colors, city scenes, clear Times Square elements.

Why Do the First Two Prompts Look So Similar And The Third Stands Out?

1. Prompt Specificity Matters

Prompt	Visual Detail	Visual Signal Strength
"at Duke University, Durham"	Low	Vague — generic location, lacks strong visual features
"at New York"	Medium	A well-known city, but no clear visual anchor
"in Times Square, ... with buildings and billboards"	High	Highly specific, with strong visual cues (Times Square, buildings, billboards)

2. Stable Diffusion's Training Data Bias

   Prompts like *"Times Square"* appear **frequently** in training data (e.g., LAION-5B), often paired with flashy, iconic images. Prompts like *"Duke University"* or even *"New York"* (by itself) may appear in a wide variety of image contexts, often with generic city or outdoor scenes.

   This causes the model to:

   a. Fall back to default imagery for less-specific prompts (e.g., cat sitting on leaves).

b. Use memorized, detailed scenes for famous locations like Times Square.

3. Prompt Embeddings from CLIP Are Similar
   Stable Diffusion uses CLIP to turn text into embeddings and the first two prompts "a aaa cat at Duke University, Durham", "a aaa cat at New York" have very similar sentence structure. In this situation, CLIP likely generates **very similar embeddings**.

   On the other hand: "a aaa cat sitting in Times Square, New York City, with buildings and billboards around" includes **distinct, visually grounded tokens** (e.g., “buildings”, “billboards”, “Times Square”). This can produce a **richer and more distinct embedding**, which affects the generated image more.

Final Summary:

Cause	**Explanation**
Generic prompt	"Duke University" and "New York" are abstract; model uses default background
Weak visual cues	Words like “Duke” or “New York” don’t have strong visual signals unless paired with context
Similar embeddings	CLIP encodes vague prompts similarly, leading to similar images
Highly specific prompt	"Times Square + buildings + billboards" activates learned, vivid visual scenes in the model

Comparision Between Different Models

Here are the results and models they use.

1. **Stable diffusion model(v1.5)**:

2. **Multimodal llm(Gpt 4o)**, I used the same initial photos as the inputs for stable diffusion models, Prompt I used was also almost the same(Please give me a picture which includes 16(2x8) subpictures, which shows cat in the pictures I gave you sitting in Times Square, New York City, with buildings and billboards around).
   I tried this way three times, the first time the result is “I wasn't able to generate the collage image due to an error in the image generation process. If you'd like, you can resend the request or modify it, and I’ll try again!”.
   Below is the result of the second time,

   In the last time, the result is like this,

Comparison between the two models:

The advantage of diffusion models lies in their more accurate and diverse image generation. However, fine-tuning the model for image generation takes about 40 minutes on an RTX 4080 GPU. On the other hand, MLLMs (Multimodal Large Language Models) have the advantage of fast image generation (around 1 minute), but the downside is poor generation quality, which manifests in the following ways: 1. Errors occur, and images fail to generate. 2. The output doesn't follow the specified format (e.g., generating a 4x3 grid instead of the requested 2x8). 3.The subject of the image is changed. 4. Although the background is more realistic (thanks to large-scale data), the subject's actions appear less realistic.

Next Steps

To improve the baseline model and broaden the scope of our investigation, we propose several concrete next steps that target fine-tuning flexibility, evaluation robustness, and additional capabilities beyond image generation.

1. Experimenting with Different Stable Diffusion Versions

While we used Stable Diffusion v1.5 as our baseline, newer versions like **v2.1** offer improved image quality, resolution handling, and richer prompt understanding. Comparing these versions will help us identify:

- **Differences in subject fidelity**: Is fidelity the same under the same fine-tuning method.
- **Changes in generalization and diversity:** Does image outputs change with fewer training steps.
- **Compatibility:** Is the model compatible with personalization techniques like DreamBooth and LoRA.
- **Training Stability**: Do the models converge faster or require more careful hyperparameter tuning?
- **Output Quality**: Evaluate whether different versions produce more realistic or coherent images.
- **Prompt Sensitivity**: Check if certain versions respond differently to subtle prompt changes.

2. Use Different Fine-Tuning Methods (e.g., LoRA)

Full fine-tuning of U-Net is computationally expensive and prone to overfitting with small datasets. To address this, we will explore **parameter-efficient tuning** strategies.

- **Low-Rank Adaptation (LoRA)**:
  LoRA adds a small set of rank-decomposed weight matrices to the model, so the training can be done with far fewer parameters. It also significantly reduces memory

consumption and training time, making it easier to iterate on new ideas or run experiments on limited hardware.

- **Textual Inversion vs. DreamBooth**: Compare these two personalization strategies. Textual Inversion modifies only token embeddings, while DreamBooth can also fine-tune parts of the U-Net. Experiment to see which approach yields better identity preservation vs. style flexibility.
- **Layer-wise vs. Full Fine-Tuning**: Instead of updating the entire U-Net, freeze certain layers and fine-tune only the higher-level blocks or attention layers. This can reduce the risk of overfitting and lower computational cost.

3. Refine Evaluation Metrics

Currently, we rely on CLIP-based text-image and image-image similarity to measure prompt alignment and subject fidelity. To gain deeper insights, we may introduce:

- **Fréchet Inception Distance (FID)**: Measures realism and diversity of generated images against a real-image distribution.
- **Learned Perceptual Image Patch Similarity (LPIPS)**: Captures perceptual similarity between generated and reference images, more sensitive than CLIP cosine.
- **Precision & Recall for Generative Models**: Measures how many generated samples fall within the manifold of real images (precision) vs. the model's coverage of the real data manifold (recall).

4. Detection of generated image

With the growing concern of the ease of generating highly realistic images. Detecting whether an image is AI-generated (as opposed to real) is crucial for areas like digital forensics, journalism, and misinformation detection.

- **Data Collection**: Assemble a dataset of real images and AI-generated images from various Stable Diffusion versions (and other models if available).
- **Model Architecture**: Use a standard CNN (e.g., ResNet) or a vision transformer (ViT) pretrained on ImageNet and fine-tune it to classify images as "AI-generated" vs. "real."

- **Feature Analysis**: Investigate whether certain pixel-level artifacts or latent features are unique to diffusion-generated images.

References

[1] N. Ruiz, Y. Li, V. Jampani, et al., “DreamBooth: Fine Tuning Text-to-Image Diffusion Models for Subject-Driven Generation,” *arXiv preprint* arXiv:2208.12242, 2022.

[2] Heusel, Martin, et al. "Gans trained by a two time-scale update rule converge to a local nash equilibrium." *Advances in neural information processing systems* 30 (2017).