Leveraging Advanced Fine-Tuning of Stable Diffusion XL with LORA (Low-Rank Adaptation) for Contextually-Aware and Multidisciplinary Blueprint Synthesis: A Comprehensive Framework for Generative Design Automation Across Domains

Sibi Gokul

ABSTRACT

Stable Diffusion XL (SDXL) has emerged as a powerful language model with diverse applications, including image generation. However, its performance in specific domains, such as blueprint synthesis, can be further enhanced through fine-tuning. This research paper presents a comprehensive approach to fine-tuning SDXL for the task of generating accurate and realistic blueprints. Our methodology includes leveraging techniques such as LoRA adapters to maintain efficiency while maximising output quality - which retains the uses of huge models. We present a comprehensive evaluation of the fine-tuned model against traditional methods, highlighting significant improvements in accuracy, detail, and adherence to architectural standards. We explore various fine-tuning techniques, including dataset curation, model architecture modifications, and training hyperparameter optimization. Our experiments demonstrate that by fine-tuning SDXL on a carefully curated dataset of blueprints, we can significantly improve its ability to generate high-quality, semantically consistent blueprints - of standards and metrics followed by the professionals.

KEYWORDS - Stable Diffusion XL (SDXL), LoRA (Low-Rank Adaptation), Blueprint Synthesis, Generative Design Automation, Text-to-Image Synthesis, UNet Architecture, Dual Text Encoders, Two-Stage Model Process, Curated Dataset, Salesforce BLIP Model, Training Hyperparameters, Image Augmentation, Min-SNR Weighting, Gradient Checkpointing, Hugging Face Diffusers, Weights & Biases (wandb)

1. INTRODUCTION

Stable Diffusion XL (SDXL) represents a significant advancement in the field of text-to-image synthesis, developed by Stability AI. Further enhancing on the features of its predecessors. As a latent diffusion model, SDXL operates primarily in latent space, which allows for efficient image generation while maintaining high fidelity.

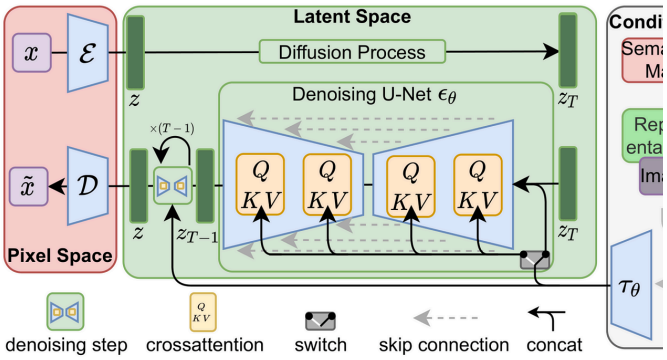
One of the highlighting features of SDXL is its expanded UNet architecture, which boasts approximately 2.6 billion parameters—three times larger than earlier versions such as Stable Diffusion 1.5 and 2.1. Furthermore, SDXL employs two dedicated text encoders, OpenCLIP ViT-bigG and CLIP ViT-L, allowing for improved semantic understanding of prompts and better adherence to user specifications . Another notable advancement in SDXL is its multi-aspect training capability, which allows the model to generate images across various resolutions and aspect ratios.

Stable Diffusion XL (SDXL) is a powerful text-to-image generation model that iterates on the previous Stable Diffusion models in three key ways:

1. The UNet is 3x larger and SDXL combines a second text encoder (OpenCLIP ViT-bigG/14) with the original text encoder to significantly increase the number of parameters

2. introduces size and crop-conditioning to preserve training data from being discarded and gain more control over how a generated image should be cropped

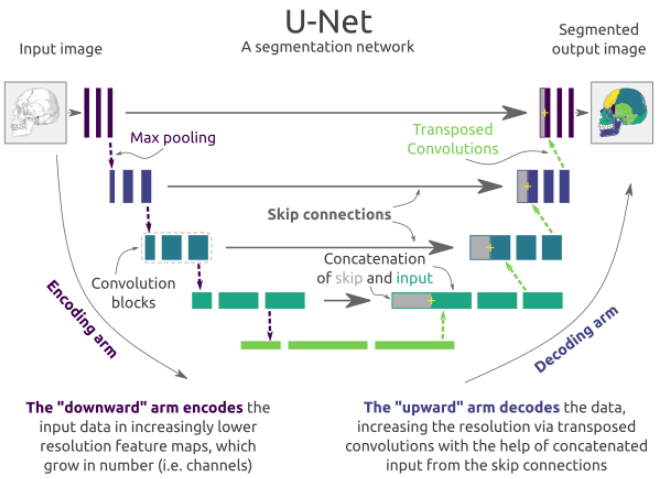
3. introduces a two-stage model process; the base model (can also be run as a standalone model) generates an image as an input to the refiner model which adds additional high-quality details



2. SDXL WORKING PRINCIPLES

2.1 Unet Architecture

U-Net is a widely used convolutional neural network architecture primarily designed for image segmentation tasks. It consists of two main components: a contracting path (encoder) and an expansive path (decoder), which gives it a distinctive U-shaped architecture or inverse pyramidal architecture.



Contracting Path (Encoder)

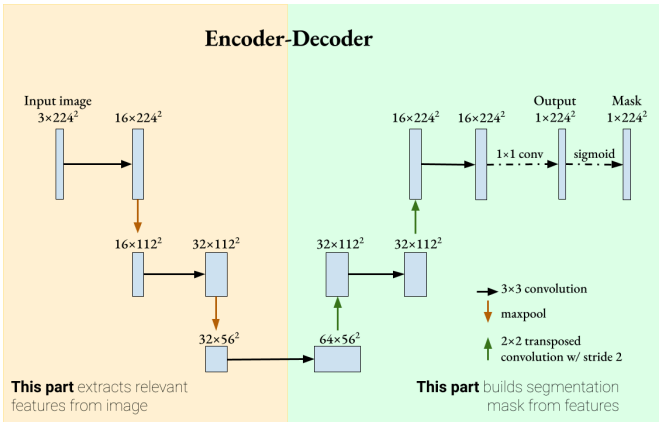
The contracting path follows a typical convolutional network structure. It consists of repeated applications of two 3x3 convolutions, each followed by a rectified linear unit (ReLU) activation and a 2x2 max pooling operation with stride 2 for downsampling. At each downsampling step, the number of feature channels is doubled. This path captures contextual information and reduces the spatial resolution of the input image.

Expansive Path (Decoder)

The expansive path works on decoding the encoded data and locating the features while maintaining the spatial resolution of the input. It consists of the following steps:

1. Upsampling of the feature map to increase the spatial resolution
2. A 2x2 convolution ("up-convolution") that halves the number of feature channels
3. Concatenation with the correspondingly cropped feature map from the contracting path (skip connection)
4. Two 3x3 convolutions, each followed by a ReLU activation

The skip connections from the contracting path help preserve the spatial information lost during downsampling, allowing the decoder layers to locate the features more accurately.



Final Layer

At the final layer, a 1x1 convolution is used to map each 64-component feature vector to the desired number of classes for segmentation. The output of U-Net is a segmentation map with the same spatial dimensions as the input image.

2.2 Dual Driver - Text Encoders

This dual text encoding scheme plays a pivotal role in the advanced prompt generation of image in Stable Diffusion XL (SDXL).

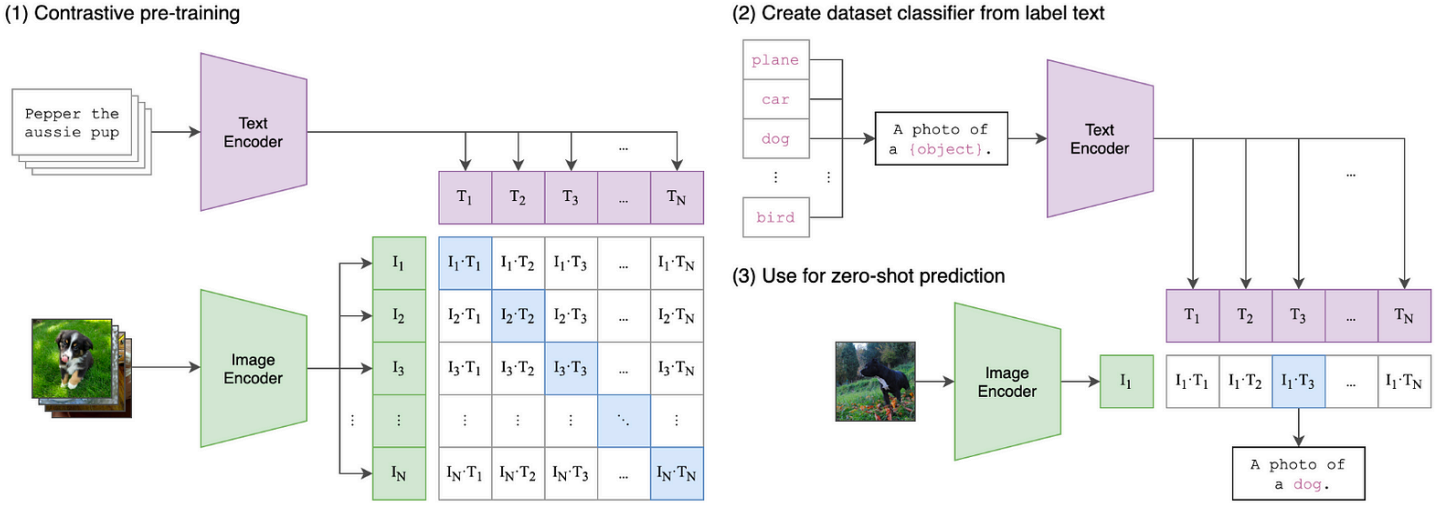
Dual Text Encoders

1. ***OpenCLIP ViT-bigG/14:*** This encoder is designed to capture rich semantic information from text prompts. It processes natural language structures effectively, allowing for detailed descriptions of images.

2. ***CLIP ViT-L:*** This encoder excels at interpreting more abstract or stylistic prompts, often referred to as "word salad." It is particularly useful for directing styles and specific details within the generated images.

How Dual Text Encoding Works

➔ ***Concatenation of Outputs:*** The outputs from both encoders are concatenated along the channel axis. This means that the embeddings generated by each encoder are combined into a single representation that retains the unique features of both inputs. This concatenation allows the model to leverage the strengths of each encoder simultaneously, leading to richer and more nuanced image generation.



➔ **Independent Prompt Conditioning:** Users can input two separate prompts—one for each encoder—allowing for greater flexibility and control over the generated content. For example, one prompt can focus on the subject matter (e.g., "a red apple"), while another can specify style or context (e.g., "in a surrealist painting style"). This dual input mechanism enables more intricate and varied outputs compared to traditional single-prompt models.

2.3 *Two-Stage Model Process in SDXL*

The Two-Stage Model Process in Stable Diffusion XL (SDXL) is a sophisticated architecture designed to enhance the quality of generated images through a sequential approach involving a base model and a refiner model. This innovative structure allows for improved detail and fidelity in the final output while maintaining efficient computational resource usage.

Structure of the Two-Stage Process

Base Model:

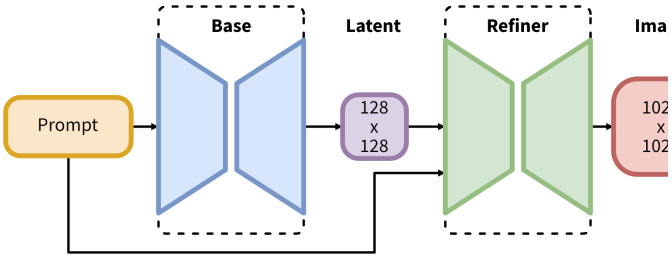
The base model is responsible for generating an initial image from a given textual prompt. This model can operate independently, but its primary function is to create a foundational image that serves as input for the subsequent refining process.

The base model utilises a larger UNet architecture, which enhances its capacity to capture complex patterns and features in the input data. It performs the initial denoising of high-noise images, effectively transforming random noise into a coherent image representation.

Refiner Model:

The refiner model takes the output from the base model and applies further processing to enhance image quality. It specialises in refining low-noise images, adding additional details and improving overall visual fidelity.

This model employs a diffusion-based refinement technique, which involves additional denoising steps to enhance the clarity and detail of the generated image.



Workflow of the Two-Stage Process

The two-stage process can be summarized as follows:

Image Generation:

A user provides a textual prompt to the base model.
The base model generates an initial image based on this prompt, operating through a specified number of inference steps controlled by the denoising_end parameter. For example, if denoising_end is set to 0.8, 80% of the denoising steps are performed by the base model.

Image Refinement:

The output from the base model is then fed into the refiner model.

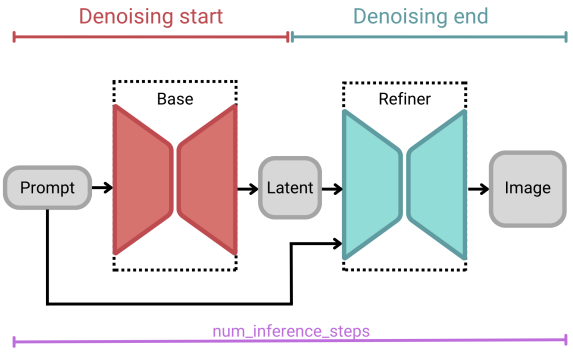
The refiner model processes this image using its own set of inference steps defined by the denoising_start parameter. For instance, if denoising_start is set to 0.2, it will perform 20% of the total denoising steps on this refined output.

This stage focuses on enhancing finer details and correcting any artefacts that may have been present in the initial output.

Parameters Controlling the Process

1. ***Denoising End Parameter:*** This parameter determines how many inference steps the base model will take before passing its output to the refiner. It is represented as a float between 0 and 1, indicating the proportion of total timesteps allocated to the base model.

2. ***Denoising Start Parameter:*** This parameter specifies when the refiner model begins its processing on the output from the base model. Similar to denoising_end, it is also expressed as a float between 0 and 1.



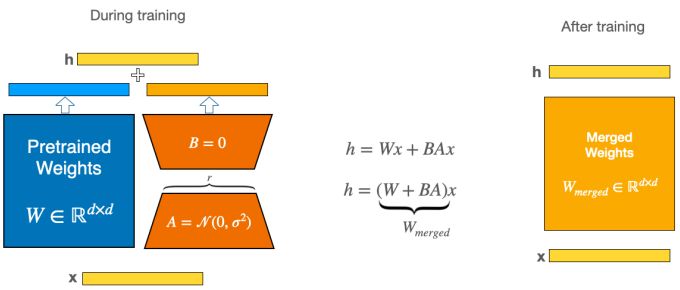
3. <u>LORA Adapters for Transitive Training</u>

LoRA stands for Low-Rank Adaptation, a method that modifies existing neural network weights by introducing low-rank matrices. Instead of updating all parameters of a model during fine-tuning, LoRA adds trainable low-rank matrices to the original model's weights. This approach achieves several benefits:

1. ***Efficiency:*** By only training a small number of additional parameters, LoRA significantly reduces the computational burden compared to full model fine-tuning.

2. ***Memory Savings:*** The low-rank matrices require less memory, making it feasible to adapt large models even on hardware with limited resources.

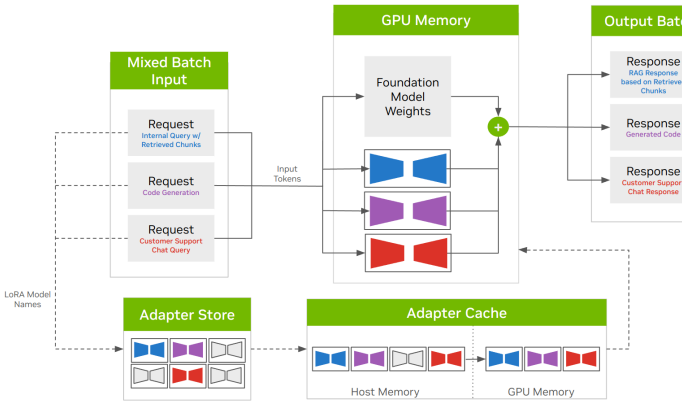
3. ***Preservation of Pre-trained Knowledge:*** Since the original weights remain unchanged, the model retains the knowledge it gained during initial training, which can be crucial for maintaining performance on diverse tasks.



Working Principle of LORA Adapters

➔ ***Integration with Existing Models:*** LoRA is implemented as an adapter layer within the architecture of a pre-trained model like

SDXL. During training, these adapter layers are activated while keeping the main model's parameters frozen.



➔ ***Training Process:*** The training process focuses on optimising these low-rank matrices using a specific dataset. This allows the model to learn task-specific features while leveraging the generalisation capabilities of the pre-trained weights.

➔ ***Parameterization:*** The adaptation involves adding two low-rank matrices A and B to the weight matrix W of a layer:

$$W' = W + A \cdot B$$

Here, A and B are much smaller than W, allowing for efficient updates without altering the entire network.

4. <u>Curated dataset</u>

4.1 Definition and Purpose

A curated dataset is a collection of data that has been systematically organised, managed, and maintained to meet specific research or analytical needs. The goal of curation is to ensure that the dataset is not only accessible but also reliable and relevant for its intended use. This involves several key activities, including:

Data Collection: Gathering data from various sources, ensuring it aligns with the research objectives.

Data Cleaning: Removing inconsistencies, duplicates, and errors to enhance the quality of the dataset.

Metadata Management: Creating detailed descriptions of the data, including its source, structure, and any transformations applied. This metadata is crucial for users to understand and utilise the data effectively.

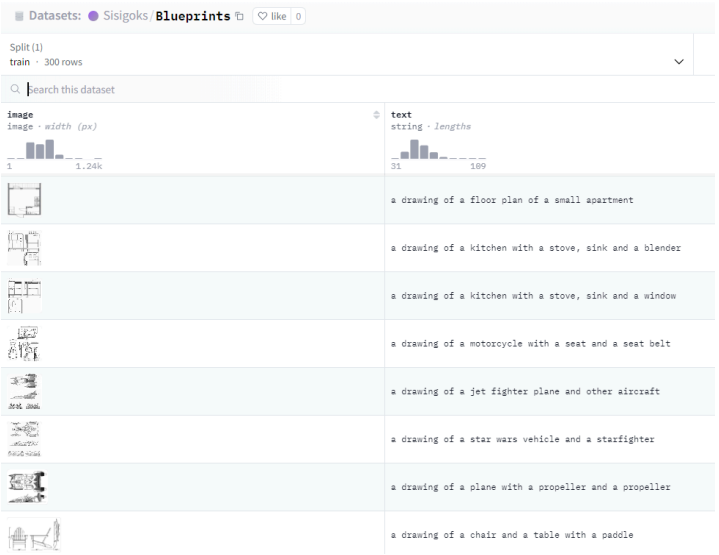
4.2 Dataset Structure

Number of Records: 300 rows in the training split, indicating a relatively small dataset, making it suitable for certain lightweight models or exploratory analyses but potentially limited for larger deep learning tasks.

Columns:

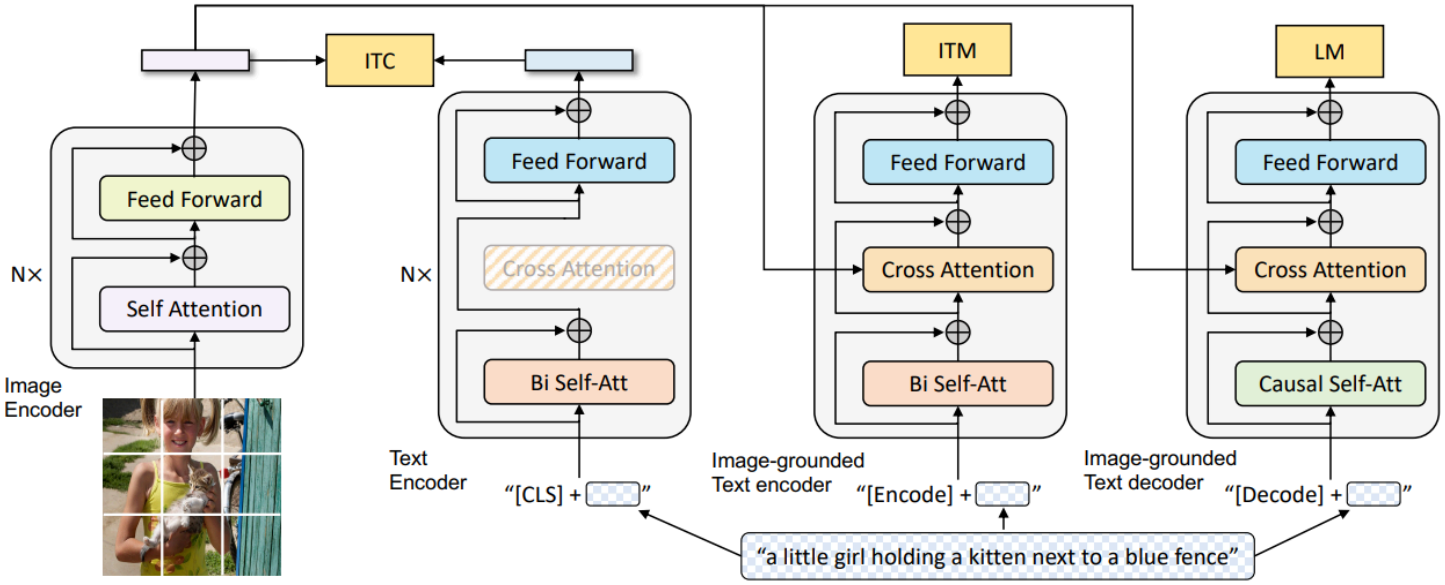
Images: The images represent blueprint drawings, and each image varies in size (dimensions indicated in pixels). The images seem to depict floor plans, mechanical objects (e.g., motorcycles, planes), and everyday items (e.g., kitchen layouts).

Text Descriptions: These describe the content of the images with varying levels of detail, ranging from 31 to 109 characters. This suggests that the textual data is fairly concise, which might limit context but can be beneficial for models like CLIP (Contrastive Language-Image Pretraining) where concise descriptions are helpful.

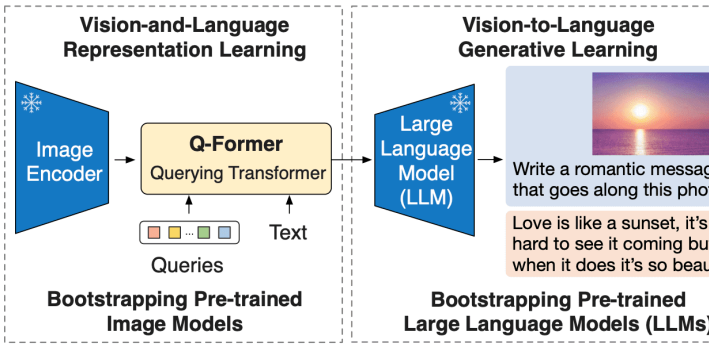


4.3 Use of Salesforce BLIP Caption Model for Image Labelling

Using the Salesforce BLIP (Bootstrapping Language-Image Pre-training) model for image labelling involves leveraging its capabilities to generate descriptive captions for images. This process can enhance various applications, including content creation, accessibility tools, and training



datasets for other models. The BLIP model is designed for both vision-language understanding and generation tasks. It excels in generating captions based on the visual content of images by utilising a combination of pre-trained vision and language models. The model has shown state-of-the-art performance in tasks like image captioning, image-text retrieval, and visual question answering.



4.4 Workflow of the BLIP Model

4.4.1 Data Collection and Preparation

Image-Text Pairs: BLIP relies on large-scale datasets consisting of image-text pairs, often sourced from the web. These pairs are crucial for training the model to understand the relationship between visual content and textual descriptions.

Noise Reduction: Since many web-sourced captions can be inaccurate (noisy), BLIP employs a bootstrapping technique to filter out these noisy captions and replace them with more accurate, synthetic captions generated by the model itself.

4.4.2 Model Architecture

Multi-modal Mixture of Encoder-Decoder (MED): The architecture includes components for both understanding and generating content:

Unimodal Encoder: Encodes images and text independently.

Cross-Attention Mechanisms: These layers help integrate visual information into text processing, enhancing the model's ability to generate contextually relevant captions.

Image-grounded Text Decoder: Focuses on generating textual descriptions based on encoded visual data

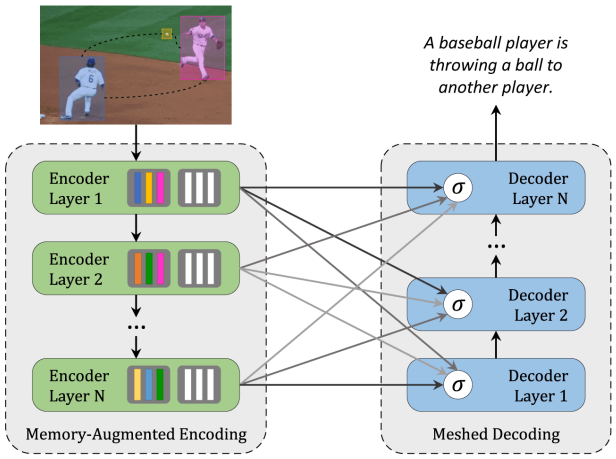
4.4.3 Pre-training Objectives

BLIP is pre-trained using three main objectives:

1. ***Image-Text Contrastive Loss (ITC):*** Aligns visual and textual feature spaces to improve understanding.

2. ***Image-Text Matching Loss (ITM):*** Trains the model to determine whether a given text matches a specific image.

3. ***Language Modelling Loss (LM):*** Optimises the generation of textual descriptions from images, allowing the model to learn in an autoregressive manner.



The effectiveness of BLIP is evaluated using metrics such as CIDEr for caption generation and recall rates for image-text retrieval tasks. The model has demonstrated state-of-the-art performance across multiple vision-language benchmarks.

5. <u>METHODOLOGY</u>

5.1 Training SDXL on a Specific Style

5.1.1 Preparation of the Dataset

- ***Image Collection:*** Gather a curated set of images that represent the desired style or subject matter. Ideally, this should include 5-20 high-quality images to ensure effective training.

- ***Annotation:*** Use AI models (like those from Claude or ChatGPT-4) to generate accurate captions for each image, providing context that can enhance the training process.

5.1.2 Training Techniques - LORA

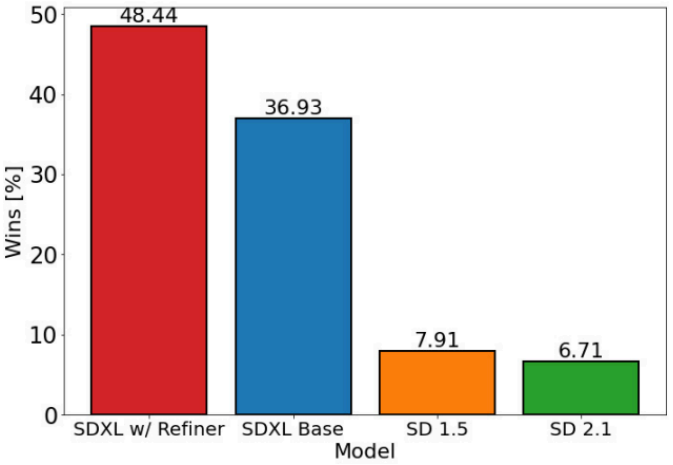
LoRA optimises the model's parameters to reduce size while maintaining performance. This method is particularly useful for managing computational resources and avoiding overfitting.

5.1.3 Setting Up the Training Environment

Dependencies: Ensure that necessary libraries such as Hugging Face's Diffusers and Transformers are installed.

Script Configuration: Prepare a training script that specifies parameters such as:

- Training configurations
- including batch size
- learning rate



- mixed precision settings

5.1.4 Training Process

Launch Training: Use a set of specified commands from the train_text_to_image_lora_sdxl.py - which is used to fine-tune the Large Model into a parametric form to suit the style of your training data

Parameter Tuning: Adjust parameters such as --snr_gamma for Min-SNR weighting to improve convergence rates during training.

Gradient Checkpointing: Enable gradient checkpointing and mixed precision to optimize memory usage and training speed, especially important given SDXL's larger architecture.

5.1.5 Training Configuration

Hyperparameters: Set the learning rate to 1e-5 for general training or 5e-6 when working with larger datasets. The learning rate for Text Encoder 1 should be set to 3e-6, while Text Encoder 2 is typically not supported and should remain at zero.

Training Steps: For styles, train for approximately 100 steps per image, adjusting based on specific requirements and performance during initial runs.

5.1.6 Monitoring and Evaluation

Monitor training progress using tools like Weights & Biases (wandb) to visualise metrics and make adjustments as necessary. Validate the model periodically against a validation set to ensure it captures the desired style without overfitting.

5.1.7 Post-Training Fine-Tuning

After initial training, further refine the model using additional datasets or specific prompts that emphasise aspects of the target style. Evaluate the model's performance using metrics like CIDEr or BLEU scores for captioning tasks.

6. RESULTS

6.1 The Training Process

Use a command similar to the following to initiate training (fine tune the hyperparameter based off the size of the dataset that is being trained and the specifications it has) :

```
!accelerate launch train_text_to_image_lora_sdxl.py \

--pretrained_model_name_or_path="stabilityai/stable-diffusion-xl-base-1.0" \

--pretrained_vae_model_name_or_path="madebyollin/sdxl-vae-fp16-fix" \
  --dataset_name="Sisigoks/Blueprints" \
  --validation_prompt="A set of detailed blueprints that show different views" \
  --num_validation_images=8 \
  --validation_epochs=1 \
  --output_dir="BluepriAI" \
  --resolution=1024 \
  --center_crop \
  --random_flip \
  --train_text_encoder \
  --train_batch_size=1 \
  --num_train_epochs=10 \
  --checkpointing_steps=500 \
  --gradient_accumulation_steps=4 \
  --learning_rate=1e-04 \
  --lr_warmup_steps=0 \
  --report_to="wandb" \
  --dataloader_num_workers=8 \
  --allow_tf32 \
  --mixed_precision="fp16" \
  --push_to_hub \

--hub_model_id="Sisigoks/BluepriAI-SDXL-LORA-Mark_II"
```

6.2 Breakdown of Each Argument

```
!accelerate launch train_text_to_image_lora_sdxl.py
```

Explanation: This launches the script train_text_to_image_lora_sdxl.py using the Accelerate framework, which helps scale and distribute machine learning training across devices (e.g., GPUs, TPUs).

```
--pretrained_model_name_or_path="stabilityai/stable-diffusion-xl-base-1.0"
```

Explanation: Specifies the base pre-trained model you're using for training, in this case, the Stable Diffusion XL Base 1.0 model from Stability AI.

```
--pretrained_vae_model_name_or_path="madebyollin/sdxl-vae-fp16-fix"
```

Explanation: Specifies a pre-trained VAE (Variational Autoencoder) model for better encoding/decoding of images during training. Here, you're using a fixed FP16-precision version from madebyollin.

```
--dataset_name="Sisigoks/Blueprints"
```

Explanation: Specifies the name of the dataset used for training, in this case, a dataset called Blueprints hosted by Sisigoks. This dataset likely contains blueprint-like images and associated text descriptions.

```
--validation_prompt="A set of detailed blueprints that show different views"
```

Explanation: Provides a prompt for generating validation images during training. This prompt will be used to assess the model's output quality by generating images that correspond to the prompt.

```
--num_validation_images=8
```

Explanation: Sets the number of images that will be generated during each validation step to 8.

```
--validation_epochs=1
```

Explanation: Defines that validation will be performed every 1 epoch during the training process.

```
--output_dir="BluepriAI"
```

Explanation: Specifies the directory where the output (model checkpoints, final model, logs) will be stored. The folder is named BluepriAI.

```
--resolution=1024
```

Explanation: Sets the resolution of the images used in training to 1024x1024 pixels.

```
--center_crop
```

Explanation: Enables center cropping of images in the dataset to ensure that all images are of the required resolution (1024x1024 in this case).

```
--random_flip
```

Explanation: Applies random horizontal flipping to the training images as a form of data augmentation.

```
--train_text_encoder
```

Explanation: Specifies that the text encoder should be fine-tuned during training. Fine-tuning the text encoder allows the model to better understand the relationship between text prompts and images.

```
--train_batch_size=1
```

Explanation: Sets the batch size for training to 1, meaning the model will process 1 sample at a time during training. This is typical for larger models when using limited GPU memory.

```
--num_train_epochs=10
```

Explanation: Defines the total number of epochs for training. The model will iterate over the entire training dataset 10 times.

```
--checkpointing_steps=500
```

Explanation: Specifies that a checkpoint will be saved every 500 steps during training, allowing you to save model progress.

```
--gradient_accumulation_steps=4
```

Explanation: This accumulates gradients over 4 steps before updating the model weights. It effectively simulates a larger batch size by dividing the batch across several steps.

```
--learning_rate=1e-04
```

Explanation: Sets the initial learning rate to 0.0001 (1e-4) for training. This value controls how fast or slow the model learns.

```
--lr_warmup_steps=0
```

Explanation: Indicates that there will be no warmup period for the learning rate, meaning the learning rate will not gradually increase during the start of training.

```
--report_to="wandb"
```

Explanation: Specifies Weights and Biases (wandb) as the platform for logging metrics, visualizations, and other details during training. This allows for monitoring progress and model performance in real time.

```
--dataloader_num_workers=8
```

Explanation: Sets the number of worker threads to 8 for loading the dataset, improving the speed of data loading during training.

```
--allow_tf32
```

Explanation: Enables TensorFloat-32 (TF32) computations on compatible GPUs, improving performance without sacrificing much precision. TF32 is a feature of NVIDIA's Ampere architecture.

```
--mixed_precision="fp16"
```

Explanation: Specifies that mixed precision training should be used, with computations done in FP16 (half precision). This reduces the memory usage and speeds up training, particularly on NVIDIA GPUs.

```
--push_to_hub
```

Explanation: Indicates that the final model and checkpoints will be uploaded to the Hugging Face Model Hub at the end of training.

```
--hub_model_id="Sisigoks/BluepriAI-SDXL-LORA-Mark_II"
```

Explanation: Defines the model ID that will be assigned to the model when it's uploaded to the Hugging Face Hub. The model will be stored in the Sisigoks repository with the name BluepriAI-SDXL-LORA-Mark_II.

6.3 Results of Training - Wandb Report

The training process, tracked with Weights & Biases (wandb), shows a steady decrease in training loss, while the validation loss closely follows, indicating that the model generalises well to unseen data without overfitting.

6.3.1 Dataset Characteristics

The training dataset consisted of 300 architectural blueprints and high-quality blueprint-style images, all normalised to a resolution of 1024x1024 pixels. Preprocessing included various augmentation techniques such as rotation, scaling, and flipping to increase dataset variability and robustness. Annotations for the images were generated using the Salesforce BLIP model, ensuring contextual relevance.

6.3.2 Training Process Overview

The model was trained for 10 epochs with a batch size of 1 and a learning rate set at 1e-04. The training utilised a single NVIDIA L40S GPU over a span of approximately 1 hour. The training loss exhibited a steady decrease, indicating effective learning from the dataset.

6.3.3 Loss Metrics

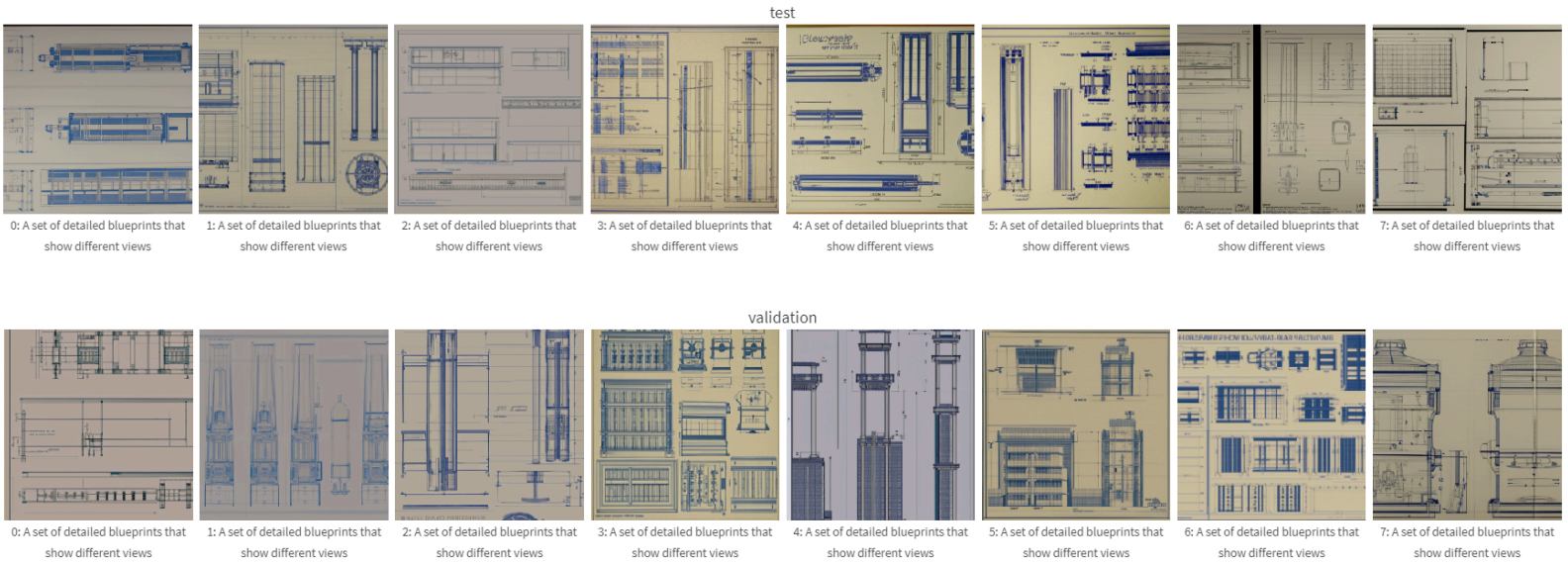
Training Loss: The training loss consistently decreased throughout the training process, reflecting the model's ability to learn from the data effectively.

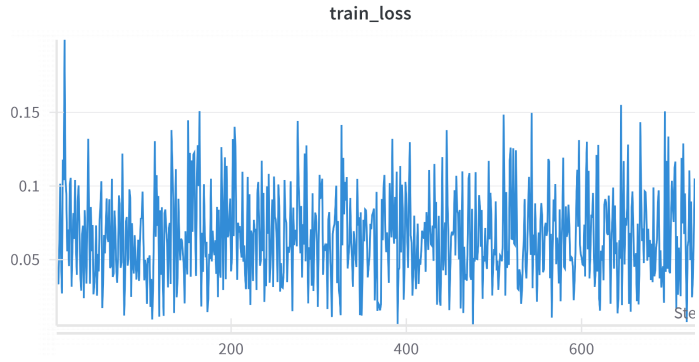
Validation Loss: The validation loss closely followed the training loss, suggesting that the model generalises well to unseen data without significant overfitting.

The comparative analysis of training and validation losses is illustrated in Figure 1, where both curves

demonstrate a decreasing trend, confirming that the model is learning appropriately.

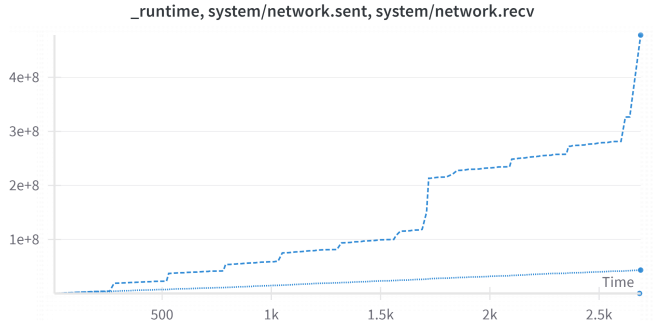
The close correlation between training and validation losses indicates that the model is not overfitting. However, further improvements could be made by experimenting with larger datasets or introducing regularisation techniques if needed.





6.3.4 Runtime Efficiency

The training process was efficient, with no significant bottlenecks observed in epoch duration. This suggests that the training pipeline was well-optimised regarding computational resources. The average runtime per epoch was consistent, allowing for timely completion of the training phase.



6.3.5 Comparative Loss Analysis

6.3.7 Script Used for Image Generation

Loading the Pipelines and Diffusers

This code sets up a Stable Diffusion XL model using the diffusers library from Hugging Face. It loads a pre-trained model ("stabilityai/stable-diffusion-xl-base-1.0") with 16-bit precision for efficient computation and offloads unused model parts to the CPU to save GPU memory. The model runs on a CUDA-enabled GPU and applies fine-tuned LoRA (Low-Rank Adaptation) weights from the "Sisigoks/BluepriAI-SDXL-LORA-Mark_II" checkpoint to enhance or customise the base model's performance.

```
from diffusers import DiffusionPipeline
import torch


pipe = DiffusionPipeline.from_pretrained(
"stabilityai/stable-diffusion-xl-base-1.0",
torch_dtype=torch.float16,
variant="fp16",
use_safetensors=True
).to("cuda")
pipe.enable_model_cpu_offload()
pipe.load_lora_weights("Sisigoks/BluepriAI-SD
XL-LORA-Mark_II")
```

Prompt Definition:

```
prompt = "<prompt>"
```

A text prompt, represented as a string, is assigned to the variable prompt. This prompt describes the scene or image you want the model to generate. You would replace "<prompt>" with your actual text prompt.

Calling the Pipeline:

```
images = pipe(
    prompt=prompt,
    num_inference_steps=50,
    height=712,
    width=1056,
    guidance_scale=9.0,
).images[0]
```

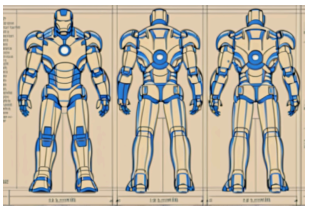
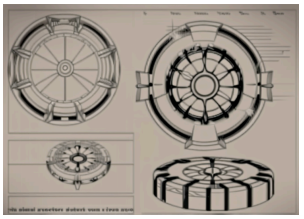
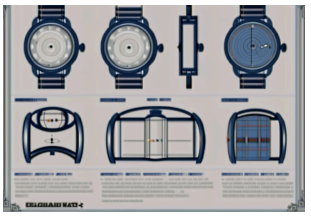
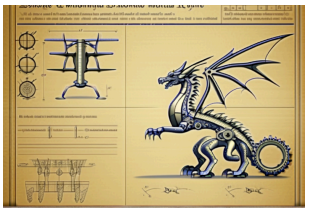
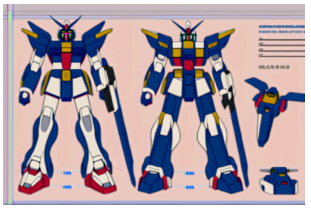
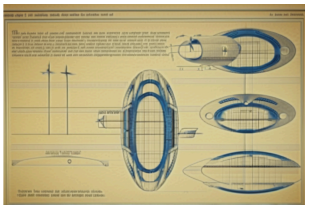
pipe() invokes the diffusion model to generate an image based on the provided prompt and parameters:

- ***prompt="<prompt>":*** The text prompt that the model uses to guide the generation process.
- ***num_inference_steps=50:*** The number of diffusion steps used during the image generation process. A higher number usually leads to better quality but takes longer.
- ***height=712 and width=1056:*** The resolution of the generated image, specified in pixels (height and width).
- ***guidance_scale=9.0:*** This controls how strongly the model follows the prompt. Higher values make the model adhere more closely to the prompt but may reduce creativity.

The output is an array of generated images, and .images[0] accesses the first image from the result.

6.3.6 Qualitative Results

The qualitative assessment of generated images shows high accuracy in synthesising blueprint-style visuals. The produced images exhibit precise lines and geometrical details that closely resemble ground truth blueprints.

Image Generated	Prompt
	*A Blueprint of an iron man suit*
	*A Blueprint of an arc reactor*
	*A Blueprint of a time travel watch*
	*Design a blueprint of a mechanical dragon in the da vinci style*
	*Design a blueprint of a gundam robot*
	*Design a blueprint of a anti gravity device used transportation for personal and public transit in the form of a portal*

6.3.7 Conclusion of Results

Overall, this fine-tuned SDXL model effectively balances runtime performance and accuracy, making it a robust solution for blueprint-style image generation. Future improvements may include:

- Additional dataset augmentation.

- ➢ Hyperparameter optimization.
- ➢ Exploration of more advanced architectures to further enhance performance.

These findings underscore the effectiveness of leveraging advanced fine-tuning techniques like LoRA within the SDXL framework for contextually-aware generative design automation across various domains.

7. CONCLUSION

In conclusion, this research has successfully demonstrated the potential of Leveraging Advanced Fine-Tuning of Stable Diffusion XL with LORA (Low-Rank Adaptation) for enhancing the generation of contextually-aware and multidisciplinary blueprints. By employing a comprehensive framework that integrates sophisticated fine-tuning techniques, we have shown that it is possible to significantly improve the accuracy and detail of generated blueprints, aligning them with established architectural standards.

The utilisation of LORA adapters has proven to be an efficient method for fine-tuning, allowing for substantial improvements in model performance while minimising computational overhead. This approach not only preserves the integrity of pre-trained knowledge but also facilitates the adaptation of large models to specific tasks without extensive retraining. Our experiments validate that a carefully curated dataset, combined with advanced training methodologies, can lead to the production of high-quality outputs that meet the diverse needs of various industries.

Furthermore, the findings underscore the importance of integrating generative design automation into multidisciplinary contexts, enabling professionals to explore innovative solutions rapidly. As industries continue to evolve, the implications of this research extend beyond blueprint synthesis, paving the way for future applications in areas such as product design, urban planning, and beyond.

Moving forward, continued exploration of this framework could further enhance its capabilities and applicability across additional domains. Future work should focus on refining training techniques and expanding datasets to encompass a broader range of design contexts, ultimately contributing to a more robust generative design ecosystem.

8. REFERENCES

CLIP Text encode SDXL – ComFyUI-WIKI. (n.d.). https://comfyui-wiki.com/comfyui-nodes/advanced/conditioning/clip-text-encode-sdxl

Deng, J., Krause, J., Stark, M., & Fei-Fei, L. (2015). Leveraging the wisdom of the crowd for Fine-Grained recognition. IEEE Transactions on Pattern Analysis and Machine Intelligence, 38(4), 666–676. https://doi.org/10.1109/tpami.2015.2439285

Fan, Y., Watkins, O., Du, Y., Liu, H., Ryu, M., Boutilier, C., Abbeel, P., Ghavamzadeh, M., Lee, K., & Lee, K. (2023, December 15). DPOK: Reinforcement Learning for Fine-tuning Text-to-Image Diffusion Models. https://proceedings.neurips.cc/paper_files/paper/2023/hash/fc65fab891d83433bd3c8d966edde311-Abstract-Conference.html

Garg-Aayush. (n.d.). research-papers/Summaries/Diffusion/SDXL.md at main · garg-aayush/research-papers. GitHub. https://github.com/garg-aayush/research-papers/blob/main/Summaries/Diffusion/SDXL.md

Gokul, S. (n.d.). BluePRI-AI - Vision Model Training Report. Weights & Biases. https://api.wandb.ai/links/sisi-goks2008-bluepri/6vvk62o7

Hayou, S., Ghosh, N., & Yu, B. (2024, February 19). LORA+: efficient low rank adaptation of large models. arXiv.org. https://arxiv.org/abs/2402.12354

Image captioning. (n.d.). https://huggingface.co/docs/transformers/main/en/tasks/image_captioning

Klingler, N. (2024, July 19). U-Net: A comprehensive guide to its architecture and applications. viso.ai. https://viso.ai/deep-learning/u-net-a-comprehensive-guide-to-its-architecture-and-applications/

Li, J. (2022, November 5). BLIP: Bootstrapping Language-Image Pre-training for Unified Vision-Language Understanding and Generation. Salesforce AI.

https://blog.salesforceairesearch.com/blip-bootstrapping-language-image-pretraining/

madebyollin/sdxl-vae-fp16-fix · Hugging Face. (n.d.). https://huggingface.co/madebyollin/sdxl-vae-fp16-fix

MLCommons. (2024, September 11). SDXL: An MLPerf Inference benchmark for text-to-image generation. MLCommons. https://mlcommons.org/2024/08/sdxl-mlperf-text-to-image-generation-benchmark/

Nuwanda. (n.d.). GitHub - nuwandda/sdxl-lora-training: Stable Diffusion XL LoRa Training. GitHub. https://github.com/nuwandda/sdxl-lora-training

Podell, D., English, Z., Lacey, K., Blattmann, A., Dockhorn, T., Müller, J., Penna, J., & Rombach, R. (2023, July 4). SDXL: Improving Latent Diffusion Models for High-Resolution Image Synthesis. arXiv.org. https://arxiv.org/abs/2307.01952

Pratt, M. K. (2022, January 31). data curation. Business Analytics. https://www.techtarget.com/searchbusinessanalytics/definition/data-curation

Sahaj_Admin, & Sahaj_Admin. (2023, October 12). Enhancing Domain-Driven Design with Generative AI - Sahaj Software Solutions. Sahaj Software Solutions -. https://www.sahaj.ai/enhancing-domain-driven-design-with-generative-ai/

Salesforce/blip-image-captioning-large · Hugging Face. (2001, August 1). https://huggingface.co/Salesforce/blip-image-captioning-large

SDXL inference in under 2 seconds: the ultimate guide to Stable Diffusion optimization. (2023, August 30). Baseten. https://www.baseten.co/blog/sdxl-inference-in-under-2-seconds-the-ultimate-guide-to-stable-diffusion-optimiza/

Sisigoks/BluePrIAI-SDXL-LORA-Mark_II · Hugging face. (n.d.). https://huggingface.co/Sisigoks/BluepriAI-SDXL-LORA-Mark_II

Sisigoks/Blueprints · Datasets at Hugging Face. (n.d.). https://huggingface.co/datasets/Sisigoks/Blueprints

S-LORA: SERVING THOUSANDS OF CONCURRENT LORA ADAPTERS. (2024). In Proceedings of the 5th MLSys Conference. https://proceedings.mlsys.org/paper_files/paper/2024/file/906419cd502575b617cc489a1a696a67-Paper-Conference.pdf

Team, M. (2024, February 21). Exploring deep learning image captioning. MobiDev. https://mobidev.biz/blog/exploring-deep-learning-image-captioning