Universiteit Leiden
The Netherlands

Opleiding Informatica

Hyperparameter Optimization of Pixel-based Diffusion Models with AutoML

Robbie Claassen

Supervisors:
Jan N. van Rijn & Inês Gomes (u. Porto)

BACHELOR THESIS

Leiden Institute of Advanced Computer Science (LIACS)
www.liacs.leidenuniv.nl

28/08/2025

Abstract

Diffusion models are a powerful tool for a wide range of applications, including the generation of novel data. To fully harness their potential, certain training settings must be carefully configured. However, identifying these settings can be time-consuming and require substantial expertise. To reduce these demands, one can employ the automated machine learning technique known as hyperparameter optimization (HPO). The central question is whether applying this optimization technique improves the quality of the output of a diffusion model. To investigate this, we implemented a pipeline that trains a diffusion model from scratch and integrates it with automated machine learning. Two hyperparameter optimization techniques were applied: random search and Bayesian optimization, and their best-found hyperparameter configurations were compared against a fixed baseline. In addition, this study will explore the application of the multi-fidelity approach, successive halving, which will be employed alongside the random search technique. The results indicate that hyperparameter optimization methods enhance the quality of the output relative to the baseline, measured using FID, by 42.5% (Bayesian optimization) and 37.0% (random search), thereby reducing the time and expertise needed to identify optimal hyperparameter settings.

Contents

1 Introduction **1**
- 1.1 Research question . . . 1
- 1.2 Thesis overview . . . 1

2 Background **2**
- 2.1 Diffusion models . . . 2
  - 2.1.1 Fundamentals . . . 2
  - 2.1.2 Training in diffusion models . . . 3
  - 2.1.3 Output evaluation . . . 3
  - 2.1.4 U-Net architecture . . . 3
- 2.2 Automated machine learning . . . 4
  - 2.2.1 Methods . . . 4
  - 2.2.2 Core components . . . 5

3 Methodology **7**
- 3.1 Pipeline . . . 7
  - 3.1.1 Diffusion model . . . 7
  - 3.1.2 Automated machine learning . . . 7
  - 3.1.3 Testing . . . 8
- 3.2 Baseline . . . 9
- 3.3 Search space . . . 9
- 3.4 Search strategy . . . 12
- 3.5 Performance evaluation . . . 13

4 Experimental setup **13**
4.1 Data . . . 14
4.2 Model architecture . . . 15
4.3 Hyperparameter optimization . . . 15
4.4 Baseline vs. hyperparameter optimization . . . 17
4.5 Multi-fidelity . . . 18
4.5.1 Two-dimensional data . . . 18
4.5.2 Image data . . . 18
4.6 Hardware . . . 18

5 Results **18**
5.1 hyperparameter optimization vs baseline . . . 19
5.1.1 Two-dimensional data . . . 19
5.1.2 Image data . . . 21
5.2 Multi-fidelity . . . 22
5.2.1 Two-dimensional data . . . 22
5.2.2 Image data . . . 23

6 Conclusions and Further Research **24**

References **25**

1 Introduction

Generative models have, over the past few years, seen a groundbreaking emergence, particularly in the context of diffusion models. Rooted in principles of non-equilibrium thermodynamics [SWMG15], these models are deployable across a wide range of fields. Thanks to their substantial capabilities, diffusion models have been successfully applied to tasks such as super-resolution, image synthesis, image generation, and many additional areas [CHIS23]. In this study, we focus specifically on pixel-based diffusion models and their use in computer vision, particularly for synthetic image generation.

Despite their substantial strength, the performance of diffusion models is highly dependent on a broad set of fixed settings, known as hyperparameters. These hyperparameters can be found both in the model's architecture and in its training process. This study focuses on the latter. Training-related hyperparameters often have a significant influence on model performance and are typically non-trivial to configure, requiring careful tuning to achieve optimal results [BWL+24].

Manually configuring hyperparameters is a time-consuming and labor-intensive process, often involving extensive trial and error and requiring substantial domain expertise. The associated search space is vast, and the interdependencies between hyperparameters are often complex and difficult to interpret. These factors pose a significant barrier for researchers who aim to fully harness the potential of such models.

1.1 Research question

Our goal in this study is to examine whether a diffusion model can generate synthetic data of higher quality through the use of the automated machine learning techniques known as hyperparameter optimization.

This leads to the following research question: *To what extent can the quality of the output of a diffusion model be enhanced through the use of the AutoML-driven optimization technique, hyperparameter optimization?*

In addition to the primary research question, we also investigate a subquestion: *To what extent can the use of a multi-fidelity approach, such as successive halving, enhance the quality of the output?*

To provide an answer to these research questions, we evaluate the output of a diffusion model trained using the best-found hyperparameter configuration obtained via hyperparameter optimization. This is compared to a baseline trained with a fixed set of hyperparameters retrieved from the original implementation. The evaluation of the output is conducted using the Fréchet Inception Distance (FID).

1.2 Thesis overview

This bachelor thesis, conducted at the Leiden Institute of Advanced Computer Science (LIACS) under the supervision of Jan N. van Rijn and Inês Gomes, presents an approach that applies

automated machine learning, specifically hyperparameter optimization, to diffusion models. The remainder of this thesis is structured as follows. Section 2 introduces the relevant background on diffusion models and automated machine learning, with a focus on hyperparameter optimization techniques. Section 3 discusses the approach implemented in this study to address the research question. In Section 4 the experimental setup is presented, followed by the results in Section 5. Section 6 concludes this study with a conclusion and suggestions for further research.

2 Background

Before examining the technical aspects of this study, it is essential to first develop a clear understanding of its two foundational components: *diffusion models* and *automated machine learning*.

2.1 Diffusion models

A model capable of generating non-existent data can be classified as a generative model. Such a model is trained to learn the underlying probability distribution of a training dataset. Once trained, it can sample from this learned distribution to generate new data points that match those in the original distribution. Diffusion models have proven to be a powerful class of generative models, achieving state-of-the-art results across multiple benchmarks. Notably, Denoising Diffusion Probabilistic Models (DDPMs), introduced by [HJA20], achieved strong results on the CIFAR-10 dataset. Besides DDPMs, there are other variants of diffusion models, such as Denoising Diffusion Implicit Models (DDIMs) [SME21], Score-based Generative Models (SGMs) [SSK+21], and Latent Diffusion Models (LDMs) [RBL+22]. These models differ in several aspects, as some are stochastic while others are deterministic, some operate in the pixel space rather than the latent space, and some rely on a denoiser network instead of a score function.

2.1.1 Fundamentals

The fundamentals of a diffusion model originate from non-equilibrium statistical physics and sequential Monte Carlo methods, first introduced by Sohl-Dickstein *et al.* (2015) [SWMG15]. Conceptually, in diffusion models, we define a two-step process: a forward process and a reverse process. In the forward process, the model gradually destroys the structure of the data distribution over a sequence of time steps. This elimination of the structure is achieved by gradually adding Gaussian noise to the data. By the final timestep, the data has lost all of its structure and resembles a Gaussian distribution of pure noise.

The opposite process, referred to as the reverse process, aims to recover the structure of the data distribution that was eliminated during the forward process. It operates in a sequence of time steps, starting with the pure noise distribution and attempting to recover the original data. Through this process, the model learns to recognize and reconstruct patterns and structures in the data [SWMG15].

Note that the previous paragraphs describe the theoretical framework underlying diffusion models, rather than the exact approach used during training. Models such as Denoising Diffusion Probabilistic

Models [HJA20] do not simulate the entire forward and reverse process as a continuous sequence during training.

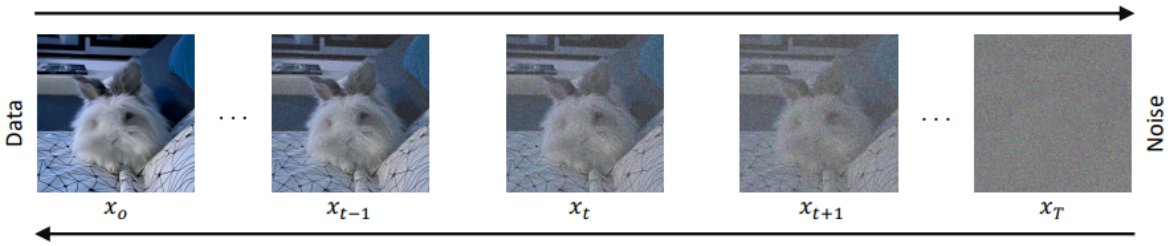


Figure 1: Forward and reverse process in diffusion models. Taken from [CHIS23].

2.1.2 Training in diffusion models

As mentioned in Section 2.1.1, the theoretical framework does not align with the approach used during the actual training of a diffusion model. Instead, an alternative training strategy is implemented. During training, a random timestep $t \in [1, T]$ is selected from the full set of timesteps. Gaussian noise is then added to a clean sample x_o, with the intensity of this noise being influenced by both the randomly selected timestep and a predefined noise schedule (e.g., linear or cosine). The resulting noisy image x_t is provided as input to the model, which is trained to predict a specific target or objective.

The learning phase of diffusion models can be guided by several commonly used training objectives. The original objective introduced by Ho *et al.* (2020) [HJA20] is known as the ϵ-prediction, where the model is trained to predict the injected noise that was added to the original clean sample.

The loss between the predicted noise and the actual injected noise is measured using the mean squared error (MSE). The aim over the course of training is to minimize this MSE; the lower the MSE, the more accurately the model becomes at predicting the training objective.

2.1.3 Output evaluation

To evaluate the generated synthetic data from diffusion models, metrics can be employed to quantify the quality of the images generated by the diffusion model. One such metric is the Fréchet Inception Distance (FID) [HRU+17]. Using this metric, features are extracted from both the real images (images from our evaluation set) and the generated samples using a pre-trained Inception V3 network. These extracted features are then modeled as Gaussian distributions. With these two distributions, we can compute the distance between the Gaussian distributions. The lower the FID score, the better the generated samples align with the evaluation set. [HRU+17]

2.1.4 U-Net architecture

The U-Net architecture is a type of convolutional neural network (CNN), and it consists of two components: a downsampling path and an upsampling path. The downsampling path decreases the spatial dimensions of an image and extracts features from the data over several steps. Each

step results in a decrease in the spatial dimension, while the number of feature maps increases. The upsampling path then, over several steps, reconstructs an image by increasing the spatial dimensions using the extracted features, to produce an output that matches the model's target objective.
We have the possibility to make several architectural design choices. In U-Nets, downsampling and upsampling operations are performed, with the number of times these operations occur depending on the number of blocks used. Each block consists, among other components, of several convolutional layers, which can likewise be configured. These layers contribute to feature detection. When configuring the number of blocks, we can also specify the number of output channels for each block. These output channels serve to store the feature maps. Naturally, the greater the number of channels, the more feature information we can store. Finally, for each block, we can define the specific type of block used for downsampling and upsampling. [RFB15]

2.2 Automated machine learning

Machine learning techniques are widely used for a wide range of applications. However, configuring these models with well-designed architectures or effective hyperparameter configurations requires sufficient expertise and involves a considerable amount of time spent on manual trial-and-error. Baratchi *et al.* (2024) [BWL+24] highlight this issue, stating that "the successes achieved by machine learning systems, however, highly rely on experienced machine learning experts who design specific machine learning pipelines" (p. 1).

Automated machine learning (AutoML) is capable of eliminating this need of expertise and manual trial and error by automating the time- and expertise-intensive process of configuring a high-performing machine learning model.

2.2.1 Methods

Automated machine learning is an umbrella term encompassing a range of techniques aimed at automating parts of the machine learning pipeline. These techniques include, among others, neural architecture search (NAS), algorithm selection, and hyperparameter optimization (HPO).

Neural architecture search (NAS) Designing an optimal architecture for a deep neural network involves selecting and tuning numerous architectural hyperparameters, such as the numbers of layers, the types of layers, connections, etc. NAS aims to automate this complex design process to identify high-performing architectures. [BWL+24, LZN+18]

Algorithm selection Selecting the most suitable algorithm for a specific task or dataset can be equally difficult and time-consuming. Originally formalized by Jon Rice in 1976 [Ric76], the concept of algorithm selection aims to identify the most effective algorithm for a given problem. [BWL+24]

Hyperparameter optimization (HPO) The primary focus of this study is hyperparameter optimization. To facilitate a better understanding of this method, it is important to clarify what hyperparameters are and distinguish them from model parameters.

- **Parameters**: A neural network consists of a large number of parameters that are updated during the training process to fit the data, however, these are not equivalent to hyperparameters.
- **Hyperparameters**: Hyperparameters, in contrast, are fixed values that remain fixed throughout training and influence how the model is trained [BWL+24].

Hyperparameter optimization involves sampling various hyperparameter configurations with the goal of discovering the best configuration that optimizes the performance of a machine learning model. From this point onward, all discussions will focus on hyperparameter optimization.

2.2.2 Core components

Three core components are central to the automated machine learning approach, and this section examines how each of them impacts the process of hyperparameter optimization [BWL+24].

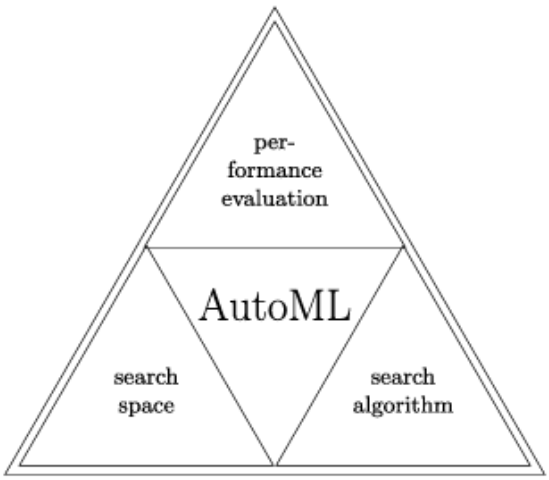


Figure 2: Core components of AutoML. Taken from [BWL+24].

Search space As previously noted, hyperparameters are predefined settings established prior to the training process, and they significantly influence how the model learns. Identifying them and their respective types (e.g., categorical, integer, continuous) is crucial. These hyperparameters, alongside their range of candidate values, collectively reside within this search space. Some may be conditional, meaning they are only relevant in the case that another hyperparameter has a specific value. [BWL+24]

Search strategies Having established a search space comprising various hyperparameters and their possible value ranges, a strategy is needed to select different hyperparameter configurations that will be applied to train a model. A search algorithm defines a specific method for the selection of these configurations. Commonly used search algorithms are: random search [BB12] and Bayesian optimization [HHL11, SLA12]. In addition to these strategies, a multi-fidelity approach such as successive halving [JT16] can be applied.

- **Random search:** Random search operates as an algorithm that, as the name suggests, randomly selects values from the search space to construct a hyperparameter configuration. It is one of the simplest search techniques for hyperparameter optimization and is frequently used as a baseline. As is evident, each selected configuration requires the full process of model training and evaluating, which can be time-consuming depending on various factors. Baratchi *et al.* [BWL+24] note that a "large number of these model evaluations are required using random search, especially with a large search space."

- **Bayesian optimization:** To reduce these model evaluations, an alternative method known as Bayesian optimization has been introduced [GH20, HHL11, SLA12, BWL+24]. Bayesian optimization leverages knowledge of its previous evaluated configurations to make more informed decisions about which configuration to explore next. Rather than randomly selecting configurations, the aim is to navigate more intelligently through the search space [BWL+24].

  There are two core components that form the backbone of Bayesian optimization. The first is the surrogate model, a predictive model trained on the outcomes of previously evaluated hyperparameter configurations. This model enables Bayesian optimization to estimate the objective function, which represents the evaluation score a trained model receives with its hyperparameter configuration. By doing so, it facilitates more informed decisions concerning which configurations to explore next [SSW+16, BWL+24].

  The second core component is the acquisition function, which leverages the information from the surrogate model to formulate a decision on what hyperparameter configuration the model should be trained and evaluated on next. This function strikes a balance between exploration and exploitation, ensuring that we make decisions on previously identified promising results (exploitation), as well as encouraging the exploration of relatively unexplored areas of the search space (exploration). [SSW+16, BWL+24]

- **Successive halving:** Although not a strict search algorithm but rather a resource allocation method, successive halving builds on the multi-fidelity concept. With multi-fidelity, we operate across multiple budget levels for training. This algorithm allocates different budgets (fidelities) during training in a manner that avoids wasting resources on underperforming configurations. Unpromising candidates can be pruned early, thereby preventing inefficient use of compute resources on models that perform worse relative to others. The core principle involves training all configurations with an initial, limited budget, followed by evaluating and ranking each candidate upon completion of this budget. Only the top fraction is selected to proceed, and this fraction is subsequently allocated an increased budget to continue training [JT16, BWL+24].

Performance evaluation As discussed above, in automated machine learning, it is essential to evaluate the various hyperparameter configurations. To achieve this, a performance evaluation metric is required, such a performance metric tool is capable of quantifying the performance of a model, trained under a specific configuration. The aim is to identify the configuration that yields the highest performance [BWL+24].

3 Methodology

The study aims to assess the feasibility of improving the output quality of a diffusion model by implementing hyperparameter optimization, an AutoML technique. The section is structured around the following key aspects: the application of AutoML to diffusion models, the construction and exploration of the search space, and the evaluation of the model's performance.

3.1 Pipeline

To address the research question, we developed a pipeline capable of training multiple diffusion models, each with a distinct configuration of hyperparameter values. This was achieved through the application of an automated machine learning framework. Each trained model instance was evaluated using a metric designed to assess the quality of the output produced by the diffusion model. All the code is available in the GitHub repository.[1]

3.1.1 Diffusion model

It is essential in this implementation to be able to train neural network diffusion models from scratch, given that each new set of hyperparameters requires complete retraining. The Hugging Face Diffusers library [vPPL+22] has made this objective highly user-friendly, providing clear and coherent instructions to facilitate its implementation. This library is accompanied by easy-to-follow documentation describing various facets relevant to diffusion models and the functioning of their library. Notably, it includes a *step-by-step guide*[2] on the process of building a pipeline capable of training a diffusion model from scratch. The model trained in this implementation adopts a U-Net architecture, as described in Section 2.1.4.

A configuration class, primarily intended for hyperparameters, is applied as part of this implementation and thus provides the capability to configure these values with minimal effort. Originally, this configuration class lacked many additional hyperparameters, which were scattered throughout the code.

3.1.2 Automated machine learning

Having established a pipeline possessing the capability of training a diffusion model with predefined hyperparameter settings, the next step involves automating the search process of identifying the best-performing hyperparameter settings. A number of libraries are available that support integrating automated machine learning strategies, such as hyperparameter optimization. One example of these libraries is Optuna [ASY+19]. Following this reference and further analysis, Optuna was selected given its ease of use and applicability for this study's task. There are a few important notations that Optuna uses in its implementation of the automated search.

Objective function As discussed earlier in the context of AutoML, the objective function plays a pivotal role in hyperparameter optimization. Optuna similarly frames the optimization task as

[1] https://github.com/wnRuppert/AutoML-DiffusionModels
[2] https://huggingface.co/docs/diffusers/tutorials/basic_training

the problem of optimizing an objective function, a function that maps a set of hyperparameter values to a performance score.

```
def objective(trial):
    hyperparameters = {
        "hyperparameter1": trial.suggest_categorical("hyperparameter1", [10,
            20, 30, 40, 50, 100, 999]),
        "hyperparameter2": trial.suggest_categorical("hyperparameter2", ["
            option1", "option2"]),
        "..."
        "hyperparameter8": trial.suggest_float("hyperparameter8", 1e-4, 3e-4,
            log=True),
    }

    train(model, hyperparameters)

    return evaluate(model)
```

Listing 1: Code snippet of the AutoML process that selects a configuration from a search space, trains the model, and evaluates it using the performance evaluation procedure

Trial Within the scope of this function, we begin by specifying the hyperparameter values, which are determined based on the chosen search strategy. Once these values have been set, the model is trained with this configuration, and ultimately a metric is computed that reflects how well the model performs. A complete instance of this process is referred to as a trial in Optuna.

Study The number of trials we wish to run in order to identify the best-performing hyperparameter settings can be determined in advance. Naturally, the more trials we run, the more compute and time are required. Once all trials have been completed, the study is considered complete. In the context of Optuna, a study refers to the entire process of the experiment. To guide the search space, we can utilize the previously discussed search strategies merely by defining them within our study object.

```
study = optuna.create_study(
    direction="minimize",
    sampler="search_strategy"
    )

study.optimize(objective, n_trials="number_of_trials")
```

Listing 2: Example of code that performs hyperparameter optimization with Optuna

3.1.3 Testing

To ensure that the full scope of the pipeline, including both the diffusion model training and the automated machine learning component, operates as expected, it was necessary to evaluate each part independently and eventually in combination. The initial aim of this study was to apply the diffusion model pipeline to image data, and the pipeline was originally built with this purpose in

mind. However, upon implementation, we concluded that training multiple models on image data using our hardware would require several hours, which is far from ideal during the testing and debugging phase. As a result, a solution had to be formulated to reduce testing time in a manner such that testing and debugging is more efficient. To address this, we opted for low-dimensional datasets, which are considerably faster to train on.

In contrast to high-dimensional image data, two-dimensional data operate within a much simpler feature space, reducing both model complexity and training cost. This simplicity allows for the use of a lightweight model architecture, such as a multilayer perceptron (MLP) [GBC16], rather than a U-Net architecture.

To support this, we incorporated an MLP architectured model into our diffusion model pipeline, using the toy-diffusion implementation by `Álvaro Barbero Jiménez`[3], with the original code adapted to meet the requirements of this study.

MLP The model architecture consists of one input layer, a series of hidden blocks, and one output layer. This represents a simple fully connected feedforward neural network, commonly referred to as a multilayer perceptron. When initializing the model, we can decide on several architecture choices. The number of input features can be defined, these represent the number of distinct values describing each data point in the dataset. For the two-dimensional data, this is set to 2, one value for the x-coordinate and one for the y-coordinate of each data point. Subsequently, the number of hidden layers (blocks) in the architecture can be determined, as well as the width of each of these hidden layers by specifying the number of units [GBC16].

3.2 Baseline

The aim of establishing a baseline is to enable a comparison between the model's performance using hyperparameter values obtained through hyperparameter optimization strategies, with a baseline that does not employ such strategies. In our experiments, we thus aim to identify the best-performing hyperparameter configuration that yields an improved output quality compared to the baseline.

3.3 Search space

The search space is a selection of all hyperparameters along with their respective ranges of candidate values. Search algorithms sample within this hyperparameter space to try different configurations. If a hyperparameter or its values are excluded, the algorithm is unable to sample and thus evaluate that particular setting. It is therefore of great importance to identify the key hyperparameters in diffusion models, along with their respective value ranges, while also considering an important trade-off:

As has become clear, each sampled hyperparameter configuration is used to train and evaluate a diffusion model, a process that is time-consuming, especially given the hardware constraints of this study. This creates the need for a search space that is sufficiently small to ensure our

[3] https://github.com/albarji/toy-diffusion

experiments remain computationally feasible, while yet broad enough to include a meaningful range of configurations.

There are numerous hyperparameters involved in the training process of a diffusion model. In this section, we introduce several of the most important ones, discussing whether they are included or excluded in our experiments, and explaining any potential conditional dependencies. We begin by discussing the hyperparameters that were excluded from tuning in either or both of the experiments.

Epochs Epochs play a significant role in determining the workload of the training process. During a single epoch, the model is trained once on the entire dataset. Therefore, increasing the number of epochs results in the model seeing the data more frequently. In these experiments, we ensure that each trial in our Optuna study undergoes an equivalent workload, meaning that each trial is exposed to the same amount of data. This allows for a fairer comparison of how the hyperparameters influence the training process while maintaining the same amount of data exposure. Thus, the number of epochs is fixed throughout our experiments.

Inference timesteps Inference timesteps influence the sampling process. The number of inference timesteps determines the number of denoising steps applied to progressively remove the noise and generate data. When the number of training timesteps, which will be discussed in the following paragraph, is substantial, using all of them during inference on image data can become time-intensive. The `Diffusers` library provides the option to select a reduced number of timesteps that are evenly spaced according to the training timesteps schedule. This hyperparameter is fixed for our image data experiments to a small value, first, to ensure that each trial has an equal computational budget for generating data, and second, to maintain feasibility given the current hardware constraints.
In the case of the two-dimensional data, the inference time is minimal. Therefore, we can simply use the selected number of training timesteps during inference, meaning that the number of inference timesteps equals the number of training timesteps.

Training timesteps In contrast to inference timesteps, training timesteps affect the training process. As previously discussed, in a diffusion model, noise is progressively added over a number of timesteps. The greater the number of timesteps, the smoother the transitions from clean data to noise. At inference time, for the image data experiments, we use a fixed number of timesteps that are evenly spaced out. To ensure that each trial during sampling observes the denoising process at identical relative positions, we also fix the number of training timesteps for these experiments. For the two-dimensional experiments, we include this hyperparameter.

Noise scheduler As explained in the background section, we know that noise is added or removed with varying intensity depending on the timestep within the diffusion process, this is governed by the noise scheduler. The type of noise scheduler determines the magnitude of noise associated with each timestep. Guo *et al.* (2015) [GLH+25] argue that the performance of a diffusion model can be affected by the type of noise schedule used, stating that "selecting an appropriate noise schedule is essential for optimizing the performance and efficiency of diffusion model training." They further conclude that there is no global noise schedule applicable to all diffusion processes, as different processes may perform better with different noise schedulers. This indicates that tuning

this hyperparameter can yield substantial benefits, and is thus selected as a tunable hyperparameter in the image data experiments. For the two-dimensional data implementation, we used the GitHub repository[4], which implemented a cosine noise schedule inspired by Nichol & Dhariwal's improved DDPM [ND21]. This implementation is adopted as a fixed noise scheduler.

The following hyperparameters are included in all experiments.

Batch size The batch size influences how frequently the model's weights are updated, as it defines the number of data points used in each training step. A larger batch size means that more data points are considered simultaneously for weight updates, resulting in fewer updates per epoch, but in return increases GPU memory usage. However, using a batch size that is too large may reduce generalization performance. A good balance must therefore be found, which is why this hyperparameter is included.

Optimizer (and weight decay) A neural-network model consists of numerous trainable parameters that are updated during training to better align with the target objective. The magnitude of these updates depends, among other factors, on the learning rate settings and the loss function. The optimizer is the algorithm responsible for performing these updates to the model's weights or parameters at each training step, with the aim of minimizing the loss. Different optimization algorithms employ distinct implementation strategies. In these experiments, we sample exclusively between Adam [KB15] and AdamW [LH19]. If the search algorithm samples AdamW, we include the option to tune the weight decay as a conditional hyperparameter. Unlike Adam, where the weight decay is tied to the gradient updates, AdamW separates the two.

Within these algorithms, each parameter in our network is updated based on an individually calculated step using information from past updates. The key distinction is that in Adam the weight decay is implemented within the gradient update for each parameter, whereas in AdamW, it is decoupled and therefore applied separately. According to Loshchilov and Hutter [LH19], this decoupling improves generalization performance.

Learning rate During the training process, the neural network's weights are continuously updated by the optimizer based on how the model performs, as measured by a loss function. The learning rate controls how much these weights are updated at the end of each training step. A larger learning rate results in larger updates, increasing the risk of overshooting optimal solutions or failing to converge, while a smaller learning rate leads to smoother adjustments over time but may be too slow to ever achieve convergence. Selecting an appropriate learning rate is difficult and thus included in the experiments. Additionally, the manner in which this learning rate evolves during training, referred to as the learning scheduler, can also influence the model's performance.

Learning rate scheduler We have just seen that learning rates that are either too high or too low can lead to issues during the training process. A learning scheduler automatically adjusts the learning rate throughout the training process. It starts with the initial learning rate and gradually modifies it, depending on the type of scheduler, until it is reduced to zero by the end of training.

[4] https://github.com/albarji/toy-diffusion

For example, the constant scheduler maintains a fixed learning rate that remains constant over the course of training.

Learning rate warm-up ratio Additionally, we may choose to configure a warm-up phase within the learning rate scheduler, which gradually increases the learning rate during the initial steps of training until it reaches the predefined initial learning rate. However, since the batch size influences the number of updates applied to the model in each epoch, and consequently the total number of scheduler steps, it is necessary to formulate a method by which the warm-up steps value is influenced according to the batch size. For example, a large batch size results in a smaller total number of scheduler steps. If the warm-up steps value is set too high, exceeding the total number of training steps, the learning rate scheduler would never exit the warm-up phase. To address this, we decided to define a tunable ratio for the warm-up steps, which is then applied to the total number of training steps.

To summarize clearly which hyperparameters are tuned in our two-dimensional data and which in our image data experiments, here is a brief overview:

Two-dimensional data: training timesteps (and therefore inference timesteps), learning rate, learning rate scheduler, learning rate warm-up ratio, batch size, optimizer, and weight decay.
Image data: learning rate, learning rate scheduler, learning rate warm-up ratio, batch size, optimizer, weight decay, and noise scheduler.

3.4 Search strategy

Within this study, we implement two search strategies to identify the optimal hyperparameter configuration using Optuna's library: random search and Bayesian optimization. In addition to the random search strategy, this study will also implement the successive halving approach together with this search strategy. We have already covered the basics of these algorithms, let us now examine how Optuna integrates these strategies.

Random search Optuna employs the `RandomSampler` to implement the random search algorithm. This approach randomly selects which configurations to try, without utilizing any past information from previous trials, unlike Bayesian optimization, which learns and leverages from past results.

Bayesian optimization Optuna adopts a Bayesian optimization approach, specifically implementing the tree-structured parzen estimator (TPE) [BBBK11]. While TPE is a type of Bayesian optimization algorithm, it uses a different underlying method compared to Gaussian process based approaches. The difference lies in what these techniques aim to model. Methods using Gaussian process aim to model the probability of a score given a hyperparameter configuration. In contrast, TPE reverses this and models the probability of a specific hyperparameter configuration given a score. To utilize this strategy, Optuna provides the `TPESampler`, which implements this specific Bayesian optimization approach.

Successive halving In Optuna, a pruner is used that can decide to terminate a trial based on an intermediate evaluation of the model. During training, the model is evaluated at specific checkpoints that can be determined in advance. In a study, each trial has a minimum number of training epochs, and a reduction factor is specified to determine how many trials may continue beyond each checkpoint. For instance, with a minimum resource of 3 and a reduction factor of 2, the checkpoints for each trial would be at epochs 3, 6, 12, 24, 48, etc. At each epoch checkpoint, the model is evaluated and compared to a ranking of the trials that have already completed that checkpoint. If the current model in the current trial ranks within the top 50%, it is allowed to continue, otherwise it is terminated. At each checkpoint in the training loop, we call `trial.report` to provide the current evaluation of the model's performance and `trial.prune` to determine whether the trial is promising enough to proceed.

3.5 Performance evaluation

A crucial aspect of this research is the ability to assign a score to each hyperparameter configuration, as these scores indicate how well a configuration performs and directly guide the trajectory of the Bayesian optimization strategy.

PRDC During the testing and debugging phase of the pipeline, it was also necessary to implement a metric. At this stage, as previously mentioned, the pipeline was applied to two-dimensional data. It was considered important to select a metric that produces a score reflecting the visual quality of the result. The closer the generated data points resemble the original data, the better the score should be. After evaluating several metrics, including the Maximum Mean Discrepancy (MMD) [GBR+12], we concluded that PRDC [NOU+20] showed the best alignment with visual quality.

PRDC, which stands for Precision, Recall, Density, and Coverage, provides four distinct scores, each corresponding to one component of the acronym [NOU+20]. In our experiments, we compute the mean of these four values to derive a single score assigned to each hyperparameter configuration.

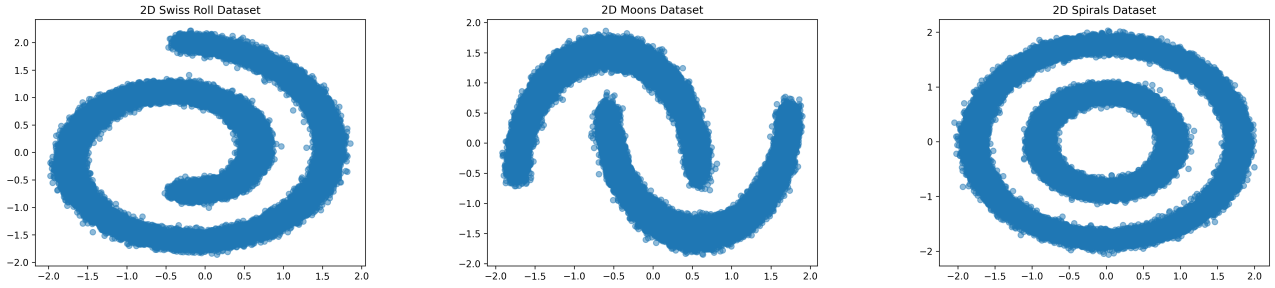
FID For our main experiments, conducted on the image data, we implemented the Fréchet Inception Distance (FID).

4 Experimental setup

As previously noted, this study employs a two-stage approach. The first stage involves testing and debugging the pipeline on two-dimensional data to enable quick prototyping, thereby facilitating a scale-up to multi-dimensional data using a fully validated pipeline. The optimization logic in both stages remains identical, making this an effective strategy to evaluate the pipeline. Accordingly, this study also includes experiments on two-dimensional data.

4.1 Data

Two-dimensional data For the two-dimensional data experiments, we use three different datasets; Swiss Roll, Moons, and Spirals. These datasets are generated with the use of three functions (make_swiss_roll, make_moons, and make_circles) from the scikit-learn library, especially the `sklearn.datasets` module [PVG+11].



(a) Swiss roll dataset (b) Moons dataset (c) Spirals dataset

Figure 3: Overview of the two-dimensional datasets.

Each dataset contains a total of 100 000 data points, of which 20 000 are used to evaluate the trained model. To ensure that performance differences are not affected by variations in the evaluation subset, each model is evaluated on the same subset.

Within these experiments, it is crucial that the data the models are exposed to is exactly the same. This ensures that both during the baseline training runs and throughout the hyperparameter optimization process, the data points remain consistent. Such consistency is essential because the goal is to assess the impact of hyperparameter optimization without the results being influenced by a model potentially being trained on more favorable synthetic data. To achieve this, all experiments on two-dimensional data are conducted using the same data points.

Image data For the image data experiments, we use the same dataset featured in the HuggingFace tutorial, namely the `smithsonian_butterflies_subset`.[5] This dataset consists of 1 000 butterfly images, of which 800 are used for training and the remaining 200 for evaluation. As the dataset contains images of varying sizes, we chose to resize all images to 64x64 pixels. The images remain sufficiently small to allow the use of the smaller U-Net architecture from our baseline in the experiments, which is highly beneficial for managing the computational constraints of this research.

[5] https://huggingface.co/datasets/huggan/smithsonian_butterflies_subset

Figure 4: Overview of five images from the butterflies dataset.

4.2 Model architecture

Two-dimensional data For the experiments on the two-dimensional data, we adopt the model architecture from the GitHub repository implementation used in our setup. As outlined in the methodology, the model is a multilayer perceptron (MLP). We configure the model with 2 input features, 4 blocks, and 64 units per block.

Image data In the case of the experiments on the image data, we adopt the U-Net architecture from our baseline. The model consists of 4 blocks, each containing two layers, with output channels of 64, 128, 128, and 256. The downsampling path is constructed of two DownBlock2D modules, followed by two AttnDownBlock2D modules. The upsampling path construction consists of two AttnUpBlock2D modules, followed by two UpBlock2D modules.

4.3 Hyperparameter optimization

For all our experiments, both on two-dimensional data and image data, we employ the two previously discussed search strategies: random search and Bayesian optimization. Both strategies are implemented using Optuna version 4.3.0. Each search strategy is allocated a computational budget of 50 trials, meaning that each strategy explores 50 different configurations within its study.

The methodology section outlined which hyperparameters were selected for tuning in our experiments. We now examine the range of values assigned to each hyperparameter, from which the search strategies can select.

Learning rate For the learning rate, we define a logarithmic search range with a minimum value of 1e-5 for both the two-dimensional and image experiments. For the image data, the maximum value is set to 1e-3, ensuring that the search space explores around the baseline value of 4e-4. For the two-dimensional data, the maximum is extended to 1e-2, reflecting the higher baseline of 1e-3. This design allows the search space to cover both lower and higher values relative to the baseline. A logarithmic range is chosen because the learning rate is sensitive to differences across orders of magnitude [ZLLS21]. The range is deliberately broad to allow exploration of both low and high learning rates.

Learning rate scheduler We use the get_scheduler function from the Diffusers library, which allows selection from several scheduler strategies. In both experiments, the options are: constant, constant_with_warmup, linear, and cosine.

Learning rate ratio (warmup steps) If the search strategy selects any scheduler other than constant, it also selects a warm-up ratio to determine the number of steps allocated to the warm-up phase. The ratio is selected from the range 0.0 to 0.2, with steps of 0.05, for both experiments.

Batch size For these values, we referred to our baseline and included one power of two below and one above it. In the case of the two-dimensional experiments, the values are 1024, 2048, and 4096. For the image data experiments, the values are 32, 64, and 128.

Optimizer (and weight decay) In the baseline for the two-dimensional data, the Adam optimizer [KB15] was used. Therefore, it is included as an option in the search space for both experiments. Additionally, we incorporated the AdamW optimizer, an improved variant of Adam introduced by Loshchilov and Hutter in [LH19], which essentially decouples the weight decay from the gradient updates. When the search algorithm samples AdamW, weight decay is included as a sampled hyperparameter. Similar to the learning rate, we define a logarithmic search range: a minimum value of 1e-6 results in a minimum regularization, with a value close to 0, while a maximum value of 1e-2 corresponds to strong regularization. This range allows us to cover a broad spectrum of the search space, from the lower bound to the upper bound.

Training timesteps The number of training timesteps for the image data experiments is fixed, as previously explained. For the other experiments it is treated as a tunable hyperparameter. The baseline uses 40 training timesteps, and our search range is centered around this value, with a minimum of 20 and a maximum of 100, with steps of 20.

Noise scheduler This hyperparameter is included only in the image data experiments. The Diffusers library provides the option to select from four noise schedulers: linear, scaled_linear, squaredcos_cap_v2, and sigmoid. All of these are included in the search space for this hyperparameter.

Table 1: Hyperparameter search space for two-dimensional experiments.

Hyperparameter	**Search space**
Learning rate	Logarithmic scale from $1 \cdot 10^{-5}$ to $1 \cdot 10^{-2}$
Learning rate scheduler	{constant, constant with warmup, linear, cosine}
Warm-up ratio	{0.0, 0.5, 0.10, 0.15, 0.20} (only if scheduler $\neq$ constant)
Batch size	{1024, 2048, 4096}
Optimizer	Adam or AdamW
Weight decay (if AdamW)	Logarithmic scale from $1 \cdot 10^{-6}$ to $1 \cdot 10^{-2}$
Training timesteps	{20, 40, 60, 80, 100}

Table 2: Hyperparameter search space for image data experiments.

Hyperparameter	**Search space**
Learning rate	Logarithmic scale from $1 \cdot 10^{-5}$ to $1 \cdot 10^{-3}$
Learning rate scheduler	{constant, constant with warmup, linear, cosine}
Warm-up ratio	{0.0, 0.5, 0.10, 0.15, 0.20} (only if scheduler $\neq$ constant)
Batch size	{32, 64, 128}
Optimizer	Adam or AdamW
Weight decay (if AdamW)	Logarithmic scale from $1 \cdot 10^{-6}$ to $1 \cdot 10^{-2}$
Noise scheduler	{linear, scaled linear, squaredcos cap v2, sigmoid}

4.4 Baseline vs. hyperparameter optimization

For all the experiments, we need to establish a baseline to enable comparison with the hyperparameter optimization results.

Two-dimensional experiments The baseline for the two-dimensional experiments is derived from the GitHub implementation[6] of the diffusion model for two-dimensional toy data. Table 3a presents this baseline.

Image experiments The baseline for the image experiments is based on the HuggingFace tutorial *Train a diffusion model*.[7] Notably, this tutorial differs from the one used to implement the diffusion model pipeline. This tutorial employs a smaller image size and, consequently, a smaller U-Net, making it computationally efficient on the available hardware. See Table 3b.

Table 3: Baseline hyperparameter values used in both experiments

(a) Two-dimensional experiments

Hyperparameter	**Value**
Learning rate	0.001
Learning rate scheduler	Linear
Learning rate ratio	0.0
Training timesteps	40
Batch size	2048
Optimizer	Adam
Weight decay	None

(b) Image data experiments

Hyperparameter	**Value**
Learning rate	0.0004
Learning rate scheduler	Constant
Learning rate ratio	0.0
Noise scheduler	squaredcos_cap_v2
Batch size	64
Optimizer	AdamW
Weight decay	0.01

For each experiment, the baseline model is trained 10 times, and the mean evaluation metric along with the standard deviation is computed. This serves as the baseline score, which we aim to surpass using our hyperparameter optimization techniques. Subsequently, the hyperparameter optimization search techniques, each consisting of 50 trials, are applied to both the two-dimensional and image

[6] https://github.com/albarji/toy-diffusion
[7] https://huggingface.co/learn/diffusion-course/unit1/2

datasets. Each search strategy finds a best-performing hyperparameter configuration, which is then used to train a model 10 times. Again, we report the mean evaluation metric and standard deviation.

4.5 Multi-fidelity

For our subquestion, we investigate whether the use of a multi-fidelity approach in hyperparameter optimization can also enhance the quality of a diffusion model's output. As described in the methodology, Optuna implements a pruner that decides during training whether a trial should be pruned based on its current performance and the performance of preceding trials. This approach allows terminating unpromising trials early, conserving computational resources for more promising trials.

In the Baseline vs hyperparameter optimization experiments, we also logged the total time it required to complete each study. In this experiment, we conduct a study using the successive halving pruner, comparing the identified best-performing hyperparameter configuration to both the baseline and the other algorithms. We also assess the total time needed to discover these configurations relative to the other algorithms. For both the two-dimensional data and image data, we apply the pruner in combination with the random search algorithm.

4.5.1 Two-dimensional data

In the experiments on two-dimensional data, we set the minimal resource to 8 epochs per trial and set a reduction factor of 2. Optuna now evaluates each trial and decides whether to prune it at the following checkpoints: epoch 8, 16, 32, 64. After the last checkpoint the trial will continue for the remaining 36 epochs.

4.5.2 Image data

In the experiments on image data, we set a minimal resource of 6 epochs per trial and a reduction factor of 2. This results in the following checkpoints where Optuna evaluates each trial and decides whether to prune: epochs 6, 12, and 24. After the final checkpoint, the trial will continue for 6 epochs to complete the total of 30 epochs for each trial.

4.6 Hardware

CPU: Intel(R) Core(TM) i7-9700K CPU @ 3.60GHz
GPU: NVIDIA GeForce RTX 5060 Ti

5 Results

In this section, we first present the results of the experiments conducted to address our main research question: whether hyperparameter optimization can improve the quality of the output of a diffusion model. We conclude by examining the results of the experiments addressing the

subquestion on successive halving, evaluating both the FID scores and the total time required for a study to complete compared to the other search strategies.

5.1 hyperparameter optimization vs baseline

In these experiments, we investigate whether the two search strategies, Bayesian optimization and random search, can give us a better hyperparameter configuration compared to the baseline. We conduct these experiments on both our two-dimensional toy dataset and the butterflies image dataset.

5.1.1 Two-dimensional data

Table 4 presents the PRDC scores (mean ± standard deviation) computed over 10 independent runs for three approaches: a baseline and two optimization strategies, random search and Bayesian optimization.

Table 4: PRDC scores (mean ± standard deviation) over 10 runs for different optimization strategies on synthetic datasets. Lower is better.

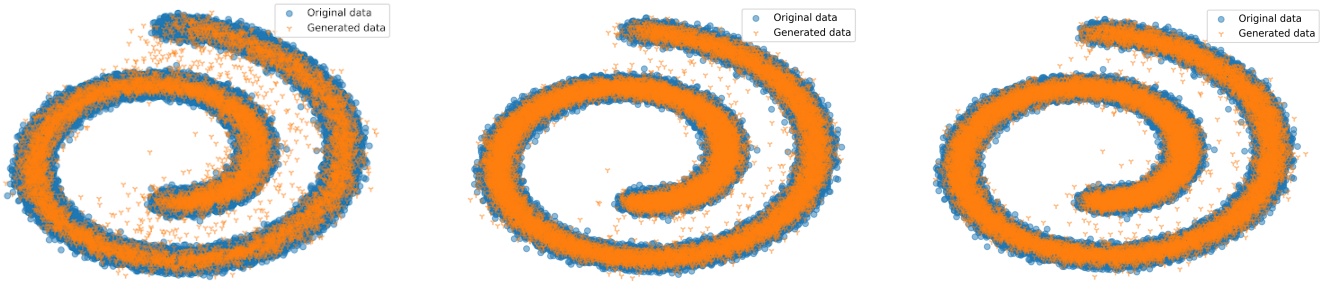
Optimization Strategy	**Swiss Roll PRDC ↓**	**Moons PRDC ↓**	**Spirals PRDC ↓**
Baseline	0.0524 ± 0.012	0.0437 ± 0.003	0.1727 ± 0.032
Bayesian Optimization	0.0166 ± 0.001	0.0131 ± 0.001	0.0186 ± 0.006
Random Search	0.0161 ± 0.001	0.0142 ± 0.001	0.0167 ± 0.002

These experiments were conducted using diffusion models trained on three 2D datasets: Swiss Roll, Moons, and Spiral. The baseline model was trained using a fixed set of hyperparameters obtained from the associated GitHub repository. For both optimization strategies, the best hyperparameter configuration identified by each method was used to train and evaluate the model across 10 separate runs. Lower PRDC scores indicate that the generated samples more closely align with the true data distribution.

As shown in Table 4, both hyperparameter search strategies outperform the baseline configuration, demonstrating that automated hyperparameter optimization can effectively improve the quality of generated outputs. Specifically:

In all visualizations shown below, the orange dots represent the data generated by the diffusion model, while the blue dots represent the original data used to train the model.

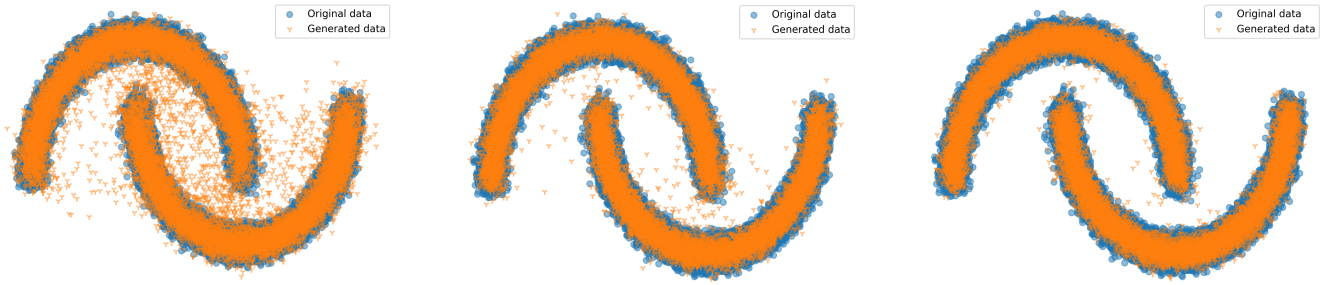
- **Swiss roll dataset:** Bayesian optimization and random search achieve a substantial reduction in PRDC score (0.0166 ± 0.001) and (0.0161 ± 0.001) compared to the baseline (0.0524 ± 0.012), indicating a closer match to the target distribution.



(a) Baseline (b) Random Search (c) Bayesian Optimization

Figure 5: Comparison of three optimization strategies

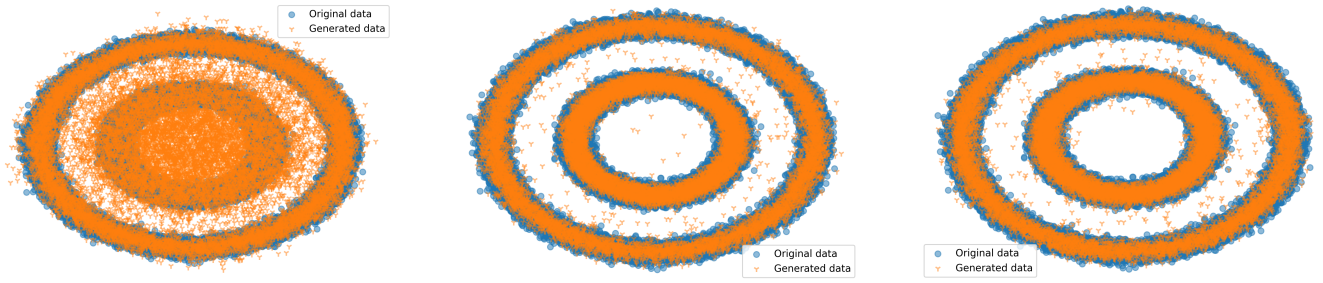
- **Moons dataset:** Bayesian optimization and random search again achieve significantly lower scores then the baseline, with the first performing slightly better (0.0131 ± 0.001).



(a) Baseline (b) Random Search (c) Bayesian Optimization

Figure 6: Comparison of three optimization strategies

- **Spiral Dataset:** On this more complex dataset, random search obtains the lowest PRDC score (0.0167 ± 0.002), and significantly outperforms the baseline configuration.



(a) Baseline (b) Random Search (c) Bayesian Optimization

Figure 7: Comparison of three optimization strategies

These results validate the effectiveness of automated machine learning techniques, such as hyperparameter optimization with different search strategies. They highlight its value as a viable approach for fine-tuning diffusion model hyperparameters, specifically in the context of low-dimensional data.

5.1.2 Image data

Table 5 presents the FID score for each approach on the Butterflies image dataset. A lower FID score indicates a better performance of the model. The scores are again computed over 10 independent runs for each of the three approaches: the baseline, Bayesian optimization, and random search. The results are reported as the mean ± standard deviation computed over these 10 independent runs.

Table 5: FID scores (mean ± standard deviation) over 10 runs for different optimization strategies on the Butterflies image dataset. Lower is better.

Optimization Strategy	**Butterflies FID ↓**
Baseline	236.79 ± 27.95
Bayesian optimization	136.17 ± 8.56
Random search	149.27 ± 5.85

The baseline model was trained using the hyperparameters provided by the hugging face tutorial. This process was repeated 10 times, and the model was evaluated in each run. For both optimization strategies, the best hyperparameter configuration identified by each method was used to train and evaluate the model across 10 separate runs.

Both search strategies achieve a lower overall FID score, indicating that they obtain more effective hyperparameter configurations compared to the baseline. We now present a selection of images generated by the models, offering visual evidence of the improvements observed, in addition to the evaluation scores.

- **Baseline**

Figure 8: Five images generated with baseline configuration

- **Random search**

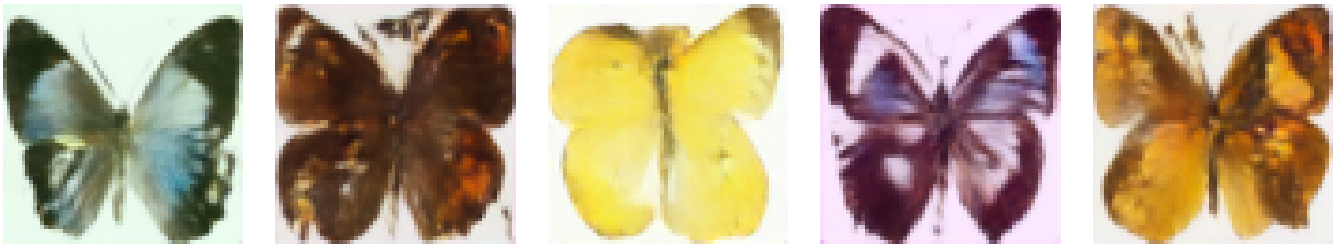


Figure 9: Five images generated with optimal configuration found with random search

- **Bayesian optimization**

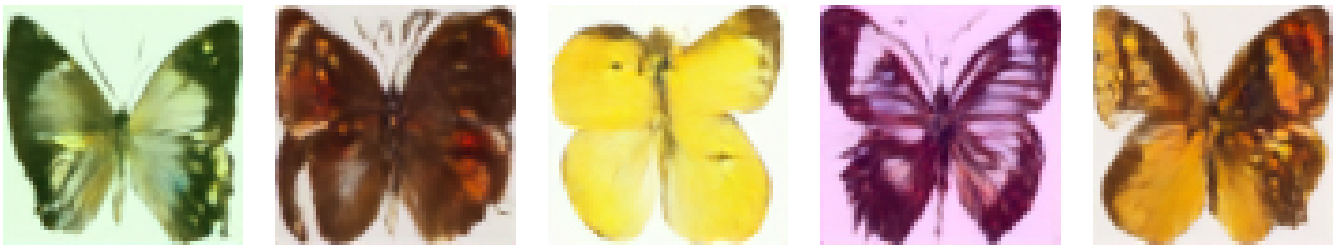


Figure 10: Five images generated with optimal configuration found with Bayesian optimization

5.2 Multi-fidelity

While conducting the previous experiments, we recorded the total time required to complete each study. We have now conducted an additional study using Optuna's pruner to demonstrate the effect of successive halving on image data, particularly to examine whether the total time decreases while still identifying configurations that enhance the quality of the output.

5.2.1 Two-dimensional data

In the following tables, we present the PRDC scores (mean ± standard deviation) over 10 runs, together with the total study time, on the swiss roll, moons, and spirals dataset.

Swiss roll

Table 6: PRDC scores (mean ± standard deviation) over 10 runs for different optimization strategies, along with the total study time.

Optimization strategy	**Swiss Roll PRDC ↓**	**Total time (minutes)**
Baseline	0.0524 ± 0.012	–
Bayesian optimization	0.0166 ± 0.001	15.49
Random search	0.0161 ± 0.001	15.23
Successive halving	0.0161 ± 0.001	3.73

Random search with and without successive halving both identified the same best-performing hyperparameter configuration, resulting in identical PRDC scores. However, the total study time required decreased from 15.23 minutes to 3.73.

Moons

Table 7: PRDC scores (mean ± standard deviation) over 10 runs for different optimization strategies, along with the total study time.

Optimization strategy	**Moons PRDC ↓**	**Total time (minutes)**
Baseline	0.0524 ± 0.012	–
Bayesian optimization	0.0131 ± 0.001	18.99
Random search	0.0142 ± 0.001	15.46
Successive halving	0.0169 ± 0.002	3.58

For the moons dataset, the random search with successive halving approach identified a different best-performing hyperparameter configuration, which was approximately 19.01% worse. However, the total study time again decreased significantly, from 15.46 minutes to 3.58 minutes.

Spirals

Table 8: PRDC scores (mean ± standard deviation) over 10 runs for different optimization strategies, along with the total study time.

Optimization strategy	**Spiral PRDC ↓**	**Total time (minutes)**
Baseline	0.0524 ± 0.012	–
Bayesian optimization	0.0186 ± 0.006	15.32
Random search	0.0167 ± 0.002	15.52
Successive halving	0.0167 ± 0.002	3.22

Similar to the swiss roll dataset, successive halving here also identified the same best-performing hyperparameter configuration as random search without successive halving, resulting in identical PRDC scores. The total study time decreased from 15.52 minutes to 3.22 minutes.

5.2.2 Image data

Table 9 presents the total time required to complete each study and reports the FID scores for each approach. Although the FID scores were already shown in Table 5, the successive halving results are now also included.

Because we ran all experiments using the same seed for the search algorithms, the random search with and without successive halving sampled the same hyperparameter configuration for each trial. The best-performing trial for the random search algorithm was trial 41, which was also identified by the random search with successive halving. This indicates that the successive halving approach

efficiently pruned trials that were less promising, thereby reducing the total compute time from 120.07 to 92.23 minutes (approx 23.2%), while still retrieving the best-performing hyperparameter configuration. In Table 9, the FID scores for random search and successive halving are identical, as both found the same configuration.

Table 9: FID scores (mean ± standard deviation) over 10 runs for different optimization strategies, along with the total study time.

Optimization strategy	**FID ↓**	**Total time (minutes)**
Baseline	236.79 ± 27.95	–
Bayesian optimization	136.17 ± 8.56	122.97
Random search	149.27 ± 5.85	120.07
Successive halving	149.27 ± 5.85	92.23

6 Conclusions and Further Research

The quality of a diffusion model's output depends heavily on its hyperparameter configuration. Establishing an effective configuration for a specific task is time-consuming and requires substantial expertise. In this study, we aimed to investigate whether the automated machine learning technique of hyperparameter optimization could assist in identifying a best-performing configuration that enhances the output quality. To explore this, we developed a pipeline capable of training a diffusion model from scratch, using the Optuna framework to explore different configurations.

As demonstrated in the results, hyperparameter optimization improved the quality of the generated outputs across all benchmarks, indicating that the model's performance can indeed be enhanced through the application of hyperparameter optimization.

On image data, random search lowered the FID from 236.79 to 149.27 (37.0%), while Bayesian optimization reduced it further to 136.17 (42.5%). Additionally, we observed that the standard deviation decreased by up to 79.1%, suggesting that the hyperparameter optimization techniques identify configurations that lead to more stable training. Across the three two-dimensional datasets (Swiss Roll, Moons, Spirals), both hyperparameter optimization search strategies, random search and Bayesian optimization, reduced the PRDC compared to the baseline. Random search achieved an average PRDC of 0.0157, reducing it by 82.5% compared to the average baseline, and Bayesian optimization achieved an average PRDC of 0.0161, reducing it by 82.0% compared to the average baseline. Furthermore, we observed that the standard deviation decreased by an average of 87.2%, suggesting that the hyperparameter optimization techniques, also for two-dimensional data, identify configurations that not only improve the output quality but also lead to more stable training.

Considering both sets of experiments, these findings suggest that the benefit of the Bayesian optimization method increases with the dimensionality of the data. This indicates that when tuning diffusion models on multi-dimensional data, such as images, applying Bayesian optimization is beneficial. For lower-dimensional data, a random search may be sufficient.

For our subquestion, we can likewise conclude that successive halving can indeed enhance the quality of the output of a diffusion model for image data, as it identifies the same hyperparameter configuration as random search without successive halving. However, in this case, it achieved this approximately 23.2% faster, reducing the time from 120.07 minutes to 92.23 minutes. This corresponds to a 1.3x speedup. For two-dimensional data, successive halving identifies the same hyperparameter configuration as without successive halving for swiss roll and spirals data. For the moons dataset, it identifies a different configuration, which still outperforms the baseline. By applying successive halving enabled, the total time required was reduced by approximately 77.0%, resulting in a 4.4x speedup across all datasets.

Despite these promising results, there was a hardware constraint that required us to conduct the image experiments on a small dataset of only 1 000 images at a resolution of 64x64, which allowed the use of a lightweight U-Net. It remains uncertain whether similar performance gains would be observed on a more diverse or multi-class dataset. Additionally, these constraints limited the number of trials, which in this study was capped at 50.

Future work For future research, the butterfly dataset used in this study, which contained only 1 000 images, could be replaced with a larger, more diverse, or multi-class dataset. Additionally, while this study focused solely on the metric score of the experiments, future work could also capture training time and GPU usage, allowing for a discussion between the trade-off of efficiency and quality.

This study demonstrates that the already powerful pixel-based diffusion model can be further enhanced through the use of AutoML techniques, substantially reducing the time and expertise required to identify a best-performing hyperparameter configuration.

References

[ASY+19] Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. Optuna: A next-generation hyperparameter optimization framework. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 2623–2631, 2019.

[BB12] James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13:281–305, 2012.

[BBBK11] James Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. Algorithms for hyper-parameter optimization. In *Advances in Neural Information Processing Systems 24: 25th Annual Conference on Neural Information Processing Systems 2011*, pages 2546–2554, 2011.

[BWL+24] Mitra Baratchi, Can Wang, Steffen Limmer, Jan N. van Rijn, Holger H. Hoos, Thomas Bäck, and Markus Olhofer. Automated machine learning: past, present and future. *Artificial Intelligence Review*, 57(5):122, 2024.

[CHIS23] Florinel-Alin Croitoru, Vlad Hondru, Radu Tudor Ionescu, and Mubarak Shah. Diffusion models in vision: A survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 45(9):10850–10869, 2023.

[GBC16] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. 2016. http://www.deeplearningbook.org.

[GBR+12] Arthur Gretton, Karsten M. Borgwardt, Malte J. Rasch, Bernhard Schölkopf, and Alexander J. Smola. A kernel two-sample test. *Journal of Machine Learning Research*, 13:723–773, 2012.

[GH20] Eduardo C. Garrido-Merchán and Daniel Hernández-Lobato. Dealing with categorical and integer-valued variables in Bayesian optimization with Gaussian processes. *Neurocomputing*, 380:20–35, 2020.

[GLH+25] Zhehao Guo, Jiedong Lang, Shuyu Huang, Yunfei Gao, and Xintong Ding. A comprehensive review on noise control of diffusion model. *CoRR*, 2025.

[HHL11] Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. Sequential model-based optimization for general algorithm configuration. In *Learning and Intelligent Optimization - 5th International Conference*, pages 507–523, 2011.

[HJA20] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems, NeurIPS*, 2020.

[HRU+17] 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. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017*, pages 6626–6637, 2017.

[JT16] Kevin G. Jamieson and Ameet Talwalkar. Non-stochastic best arm identification and hyperparameter optimization. In *Proceedings of the 19th International Conference on Artificial Intelligence and Statistics*, volume 51 of *JMLR Workshop and Conference Proceedings*, pages 240–248, 2016.

[KB15] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *3rd International Conference on Learning Representations*, 2015.

[LH19] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *7th International Conference on Learning Representations*, 2019.

[LZN+18] Chenxi Liu, Barret Zoph, Maxim Neumann, Jonathon Shlens, Wei Hua, Li-Jia Li, Li Fei-Fei, Alan L. Yuille, Jonathan Huang, and Kevin Murphy. Progressive neural architecture search. In *Computer Vision - ECCV 2018 - 15th European Conference*, pages 19–35, 2018.

[ND21] Alexander Quinn Nichol and Prafulla Dhariwal. Improved denoising diffusion probabilistic models. In *Proceedings of the 38th International Conference on Machine Learning*, pages 8162–8171, 2021.

[NOU+20] Muhammad Ferjad Naeem, Seong Joon Oh, Youngjung Uh, Yunjey Choi, and Jaejun Yoo. Reliable fidelity and diversity metrics for generative models. In *Proceedings of the 37th International Conference on Machine Learning*, pages 7176–7185, 2020.

[PVG+11] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake VanderPlas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, and Edouard Duchesnay. Scikit-learn: Machine learning in python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[RBL+22] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2022*, pages 10674–10685, 2022.

[RFB15] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *Medical Image Computing and Computer-Assisted Intervention 18th International Conference*, pages 234–241, 2015.

[Ric76] John R. Rice. The algorithm selection problem. *Advances in Computers*, 15:65–118, 1976.

[SLA12] Jasper Snoek, Hugo Larochelle, and Ryan P. Adams. Practical bayesian optimization of machine learning algorithms. In *Advances in Neural Information Processing Systems 25: 26th Annual Conference on Neural Information Processing Systems*, pages 2960–2968, 2012.

[SME21] Jiaming Song, Chenlin Meng, and Stefano Ermon. Denoising diffusion implicit models. In *9th International Conference on Learning Representations*, 2021.

[SSK+21] Yang Song, Jascha Sohl-Dickstein, Diederik P. Kingma, Abhishek Kumar, Stefano Ermon, and Ben Poole. Score-based generative modeling through stochastic differential equations. In *9th International Conference on Learning Representations*, 2021.

[SSW+16] Bobak Shahriari, Kevin Swersky, Ziyu Wang, Ryan P. Adams, and Nando de Freitas. Taking the human out of the loop: A review of bayesian optimization. *Proceedings of the IEEE*, 104(1):148–175, 2016.

[SWMG15] Jascha Sohl-Dickstein, Eric A. Weiss, Niru Maheswaranathan, and Surya Ganguli. Deep unsupervised learning using nonequilibrium thermodynamics. In *Proceedings of the 32nd International Conference on Machine Learning*, pages 2256–2265, 2015.

[vPPL+22] Patrick von Platen, Suraj Patil, Anton Lozhkov, Pedro Cuenca, Nathan Lambert, Kashif Rasul, Mishig Davaadorj, Dhruv Nair, Sayak Paul, William Berman, Yiyi Xu, Steven Liu, and Thomas Wolf. Diffusers: State-of-the-art diffusion models. https://github.com/huggingface/diffusers, 2022.

[ZLLS21] Aston Zhang, Zachary C. Lipton, Mu Li, and Alexander J. Smola. Dive into deep learning. *CoRR*, 2021.