FeatureFusion: Merging Diffusion Models Through Representation Correlations

Murdock Aubry
University of Toronto
Department of Computer Science
murdock@cs.toronto.edu

James Bona-Landry
University of Toronto
Department of Mathematics
james.bonalandry@mail.utoronto.ca

Abstract

We introduce *FeatureFusion*, a novel framework for merging diffusion models trained on different tasks without additional training. While existing methods like Diffusion Soup rely on simple linear interpolation, which works well for models trained on the same task, our approach leverages representation correlations across models to enable merging of specialists. Building upon ideas from *ZipIt!* [21], our method forms convex combinations of all model weights scaled by their correlation strength and relative operational scale, allowing diverse models to be integrated more effectively. We provide theoretical analysis of the sampling distribution of merged models and demonstrate through experiments with object-based and stylistic specialists that *FeatureFusion* preserves the capabilities of constituent models while maintaining the computational efficiency of a single model. Qualitative and quantitative results show that our approach successfully combines the strengths of specialist models trained on different tasks into a unified model with no additional inference overhead.

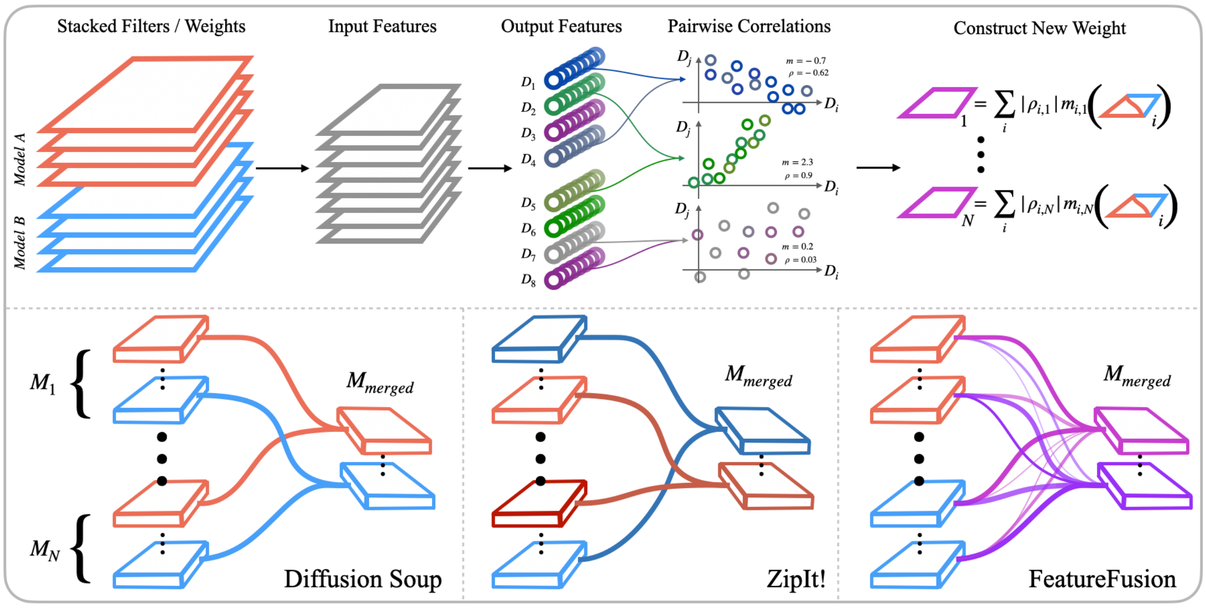

Figure 1: **Main Methodology.** A qualitative comparison of the Diffusion Soup [3] and ZipIt! [21] merging frameworks against the proposed *FeatureFusion*. The top row summarizes the process of merging linear weights and convolutional layers through the latter. The colours describe the way each method matches distinct convolutional filters and weight matrices. Diffusion Soup linearly interpolates the constituent model weights according weight names. ZipIt! pairs weights based on how correlated the corresponding features are, but only merges pairwise and with even weightings. *FeatureFusion* considers all pairwise correlations, and each new weight is a convex sum of all other weights in the corresponding layer, weighted based on the strength and scale of the correlation.

Contents

1. Introduction

Merging neural networks without re-training and without introducing additional computational overhead is a challenging task that has garnered much interest in recent years. While sophisticated approaches have been explored in the general case [21], the specific problem of merging diffusion models remains largely unexplored beyond linear interpolation of weights [3]. This approach is effective for models trained on the same task, but inadequate for models trained on different tasks, among other limitations. In this work, we propose a new approach to merging generative models by extending ideas from existing work [3, 21]. In particular, we aim to go beyond linear interpolation and achieve a more effective framework that bypasses many existing challenges by leveraging feature similarities between models, drawing inspiration from the techniques of [21].

2. Background

2.1. Diffusion Models

Diffusion models [6, 10, 19, 15, 14, 17, 18] are a class of generative models designed for generalizing on text-to-image data, and operate by iteratively refining random noise samples through a learned denoising process. During training, the model is exposed to a series of images $x_0 \sim p_{\text{data}}(x)dx$ that are gradually corrupted into pure noise $\mathcal{N}(0, I)$ according to the iterative forward process

$$x_t = \sqrt{1-\beta_t}x_{t-1} + \sqrt{\beta_t}\epsilon_t, \quad t \in \{1, \dots, T\}, \tag{1}$$

where $\epsilon_t \sim \mathcal{N}(0, I)$ is Gaussian noise and independent of ϵ_s for $s \in \{1, \dots, t-1\}$, and $\beta_t > 0$ is a small constant controlling the strength of the corruption at the current time step t. This forward process can be understood through the lens of stochastic calculus [20]. In particular, note that since β_t is small, Taylor expansion implies that $\sqrt{1-\beta_t} \approx 1 - \frac{1}{2}\beta_t$ and hence we can think of (1) as expressing that

$$x_t - x_{t-1} \approx -\frac{1}{2}\beta_t x_{t-1} + \sqrt{\beta_t}\epsilon_t, \quad t \in \{1, \dots, T\}, \tag{2}$$

which is a discretization of the diffusion equation

$$dx_t = -\frac{1}{2}\beta_t x_t dt + \sqrt{\beta_t}dB_t, \quad 0 \le t \le T. \tag{3}$$

It is a remarkable fact due to Anderson [2] that the reverse of a diffusion process is also a diffusion process. In particular, for x_t evolving according to the equation above, the time-reversed process x_s for $s = s(t) := T - t$ satisfies

$$dx_s = \left(-\frac{1}{2}\beta_s x_s - \frac{1}{2}\nabla \log p_s(x_s)\right) ds + \sqrt{\beta_s}dB_s, \quad 0 \le t \le T, \tag{4}$$

where p_s denotes the Lebesgue-density of the distribution of the latent x_s. The significance of this is that in order to simulate the reverse process and generate samples of p_{data} from pure noise, one simply needs access to the so-called *score function* $\nabla \log p_t$. In general, determining this function analytically is intractable, so in practice one instead approximates it via a neural network $s_\theta(x, t)$. Specifically, observe that by iterating (1) we can re-express the forward process as

$$x_t = \sqrt{\alpha_t}x_0 + \sqrt{1-\alpha_t}\epsilon, \quad t \in \{1, \dots, T\}, \tag{5}$$

where $\alpha_t = \prod_{s=1}^{t}(1-\beta_s)$ and $\epsilon \sim \mathcal{N}(0, I)$. It follows from this that, conditional on x_0, the latents x_t are distributed according to $\mathcal{N}(\sqrt{\alpha_t}x_0, (1-\alpha_t)I)$. One can thus write down an explicit formula for the Lebesgue-density $p_{t|0}$ of this conditional law, which can then by logarithmically differentiated to obtain the explicit formula $\nabla \log p_{t|0}(x) = -\frac{x - \sqrt{\alpha_t}x_0}{1-\alpha_t}$ for the *conditional score* $\nabla \log p_{t|0}$. In particular, note that when evaluated at the latent x_t, (5) implies that the right-hand side reduces to $-\epsilon/\sqrt{1-\alpha_t}$. One can thus train $s_\theta(x,t)$ according to the loss function

$$\mathbb{E}_{x_0\sim p_{\mathrm{data}},\epsilon\sim\mathcal{N}(0,I)}\left\|s_\theta(x_t,t)+\frac{\epsilon}{\sqrt{1-\alpha_t}}\right\|^2 \tag{6}$$

to approximate the *conditional score* $\nabla\log p_{t|0}$, and the true score $\nabla\log p_t$ can then be recovered by averaging over all possible initial states x_0, weighted according to their likelihood given x_t, that is,

$$\nabla\log p_t(x)=\mathbb{E}_{x_0\sim p_{0|t}}[\nabla\log p_{t|0}(x)]\approx\mathbb{E}_{x_0\sim p_{0|t}}[s_\theta(x,t)]. \tag{7}$$

For a more detailed and complete treatment of diffusion models through the lens of stochastic calculus, we refer the reader to [20].

2.2. Stable Diffusion

It has become standard practice to use a U-Net as the underlying neural network $s_\theta(x,t)$ in a diffusion model. They operate by encoding images into a latent space, where they are then denoised step-by-step before being decoded back into pixel space. The encoder and decoder both function through convolutional layers, which allow the input data to be reliably downsampled and upsampled by capturing relevant spatial information. In addition, timestep embeddings are injected to modulate the denoising process and guide the model from general image-defining strokes towards careful and precise refinements. Similarly, text embeddings and positional embeddings are used to condition the denoising process on the prompt and retain spatial information throughout the process. Attention blocks are sometimes also incorporated to improve overall performance through their ability to capture spatial relationships within the input data.

3. Related Work

3.1. ZipIt! [21]

Much of the current work on the merging of neural networks [1, 24, 3] centres on permutation-based interpolation approaches that require the models being merged to have been trained on the same task. While merging models in this context has its merits (e.g. increased model robustness, improved generalization, and better overall performance [22, 13]), it would be nice to have a framework for merging models trained on *different tasks* into a single model capable of performing them all, ideally without any additional computational overhead. In their 2024 paper *ZipIt! Merging Models from Different Tasks without Training* [21], Stoica et al. present a framework addressing precisely this challenge, dubbed *ZipIt!*. The basic idea is to interpolate neurons only if they have feature activations that are highly correlated. Neurons that correspond to dissimilar feature activations, on the other hand, are left separate so as to preserve task-specific information in the underlying models.

More precisely, suppose that we have two models A and B with exactly the same architecture (but not necessarily trained on the same task) that we wish to merge. If $\mathcal{L}_i^A$ and $\mathcal{L}_i^B$ are corresponding linear layers in these models with weight matrices W_i^A and W_i^B, biases b_i^A and b_i^B, and output features f_i^A and f_i^B, then they are merged according to

$$W_i^{\text{merged}}=M_i^AW_i^AU_{i-1}^A+M_i^BW_i^BU_{i-1}^B \tag{8}$$

and

$$b_i^{\text{merged}}=M_i^Ab_i^A+M_i^Bb_i^B. \tag{9}$$

Here, $M_i=(M_i^A,M_i^B)$ is a *merge matrix* that is determined by the pairwise correlations of the entries of the concatenated feature vector (f_i^A,f_i^B), and $U_i=(U_i^A,U_i^B)$ is the pseudo-inverse of M_i (referred to as the *unmerge matrix*). Post-multiplication by U_{i-1} in (8) is necessary to ensure that the outputs of layer $i-1$ get fed into the weight matrices W_i^A and W_i^B in a form that is consistent with what they learned to expect during training. It is not necessary in (9) since the biases do not interact with data from previous layers. See Section 4 of [21] for more details.

For nonlinear layers (e.g. BatchNorm, ReLU, etc.), one merges by “propagating” forward the merge and unmerge matrices M_i and U_i of the most recent preceding linear layer. What exactly this means differs depending on the type of nonlinear layer encountered, but for convolutional layers (i.e. what one would likely see in a diffusion model) it amounts to applying M_i and U_i to each kernel location in a manner analogous to (8). See the appendix in [21] for more details.

While *ZipIt!* serves as an effective and unprecedented framework for merging neural networks trained on different tasks and of all varieties, certain limitations remain, for instance:

1. **Data dependence:** in order to determine the pairwise correlations of the entries of the concatenated feature vector (f_i^A, f_i^B), *ZipIt!* computes the correlations over a small pre-determined set of sample inputs. The way models are merged is therefore dependent upon the set of sample inputs used, which could possibly lead to poor generalization in models merged via this framework.
2. **Challenges with nonlinear layers:** the way *ZipIt!* handles nonlinear layers, while general, is prone to loss of information. For instance, in the case of convolution layers, there is no reason to expect that the filter weights are correlated in a way that would allow them to transform meaningfully through “propagation” of the merge and unmerge matrices, as these are tailored for a preceding linear layer and not the convolutional layer itself. The same problem is faced by other nonlinear layers.
3. **Limited experimental validation:** so far, *ZipIt!* has only been tested on small image classification models (see Section 5 of [21] for details). It is thus unclear how the framework will scale for larger and more complicated architectures, such as transformers or diffusion models, which typically involve more complex feature interactions.

The first two issues specifically are challenges that we endeavour to address in our new framework.

3.2. Diffusion Soup [3]

In their 2024 paper *Diffusion Soup: Model Merging for Text-to-Image Diffusion Models*, Biggs et al. present a framework for merging diffusion models specifically that leverages the linear mode connectivity often observed in models trained on the same task [8, 7]. Given N diffusion models (of the same architecture) with parameters $\theta_1, \ldots, \theta_N$, a merged model is constructed by “souping” the parameters θ_i according to a convex combination

$$\theta_{\text{soup}} := \sum_{i=1}^{N} \alpha_i \theta_i, \tag{10}$$

where the coefficients α_i are hyperparameters to be optimized prior to inference. The authors present two different algorithms to this effect, both greedy algorithms based on iteratively updating the α_i according to performance on a pre-determined set of test data. See Section 4.2 of [3] for more details.

The authors go on to show that, with interpolation coefficients optimized in this fashion, a *Diffusion Soup* of specialist models trained on different data shards outperforms a monolithic model trained on the union of these data shards, specifically according to the evaluation metrics of IR and TIFA score (see Section 6 of [3] for details). Theoretically, they attribute these findings to the fact that a *Diffusion Soup* of models (approximately) samples from the geometric mean of the sampling distributions of the constituent models, whereas a monolithic model (approximately) samples from the arithmetic mean of these distributions (see Section 4.1 of [3] for details). The idea is that, by definition, the geometric mean emphasizes regions in data space where all the constituent models agree (in the sense of $\mathbb{P}_i(x)$ being high for all i, where $\mathbb{P}_i(x)$ denotes the likelihood of the image x according to the sampling distribution of

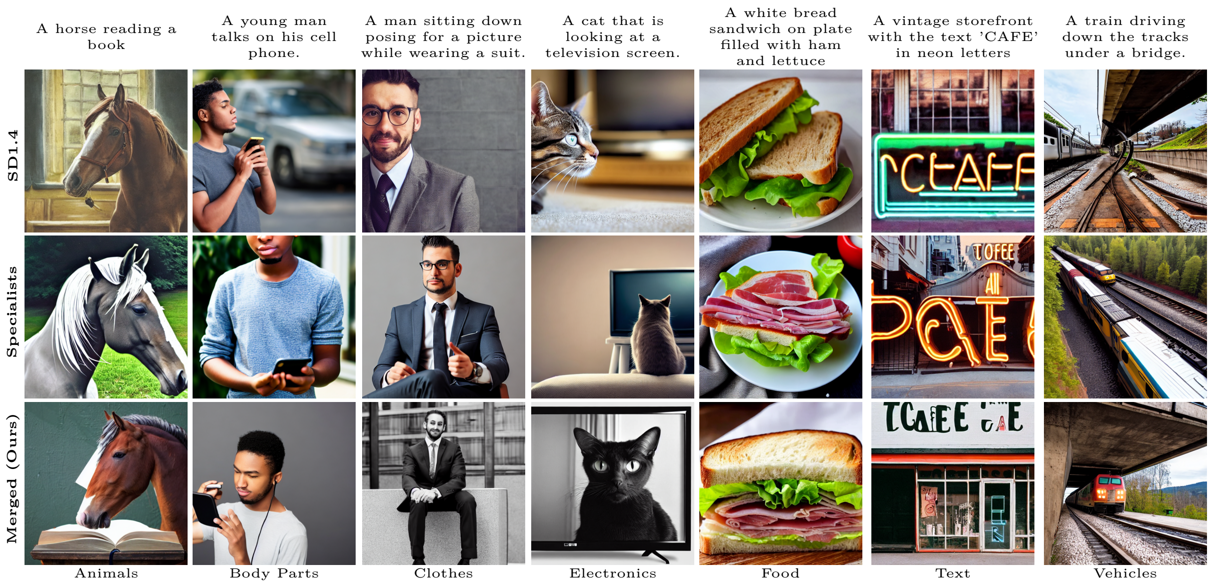
A horse reading a book
A young man talks on his cell phone.
A man sitting down posing for a picture while wearing a suit.
A cat that is looking at a television screen.
A white bread sandwich on plate filled with ham and lettuce
A vintage storefront with the text 'CAFE' in neon letters
A train driving down the tracks under a bridge.
SD1.4
Specialists
Merged (Ours)
Animals
Body Parts
Clothes
Electronics
Food
Text
Vehicles

Figure 2: **Object-based Qualitative Results.** Sample outputs of the base SD1.4 model, the specialist models, and the cumulative merged model on various image generation tasks. The specialist images are tested on prompts which align with the respective specialty. The merged outputs are the results of the merging of the seven specialists with the based SD1.4 model.

model i), and suppresses regions in data space where they disagree. This is in stark contrast with the arithmetic mean, which is sensitive to outliers in the data shards.

The main limitation of the *Diffusion Soup* framework, however, is that it requires the models being merged to have been trained on the same tasks. This is one of the key issues that we address in our new framework by drawing inspiration from the techniques of *ZipIt!* [21].

4. New Approach: FeatureFusion

4.1. Overview

Consider N diffusion models with identical architectures: a "base" model M_1 and $N-1$ specialist models $M_2, \ldots, M_N$. Our goal is to merge these models into a single model M_{merged} which combines the capabilities of each at no additional computational overhead. The framework we present here operates on a layer-by-layer basis (i.e. we construct the merged model layer-by-layer by combining the corresponding layers of the constituent models) and works exactly the same for both linear layers and convolutional layers (i.e. what one would find in the U-Nets underlying the constituent models). For explanatory clarity, we use the example of convolutional layers in the following sections detailing the merging procedure. For linear layers, one should replace every instance of "filter" with "weight matrix", and every instance of "subfilter" with "row of a weight matrix".

4.2. How do we merge convolutional layers?

In the *ZipIt!* framework, the core idea of merging based on feature correlation is applied only to linear layers. For nonlinear layers (e.g. convolutional layers), merging is instead achieved by "propagating forward" the merge and unmerge matrices of the most recent preceding linear layer. As previously discussed, this is a lossy operation because there is no reason to expect that the filter weights of a given convolutional layer are correlated in a way that would allow them to transform meaningfully under the merge and unmerge matrices of a preceding linear layer. The new approach we present here offers a solution to this problem by generalizing the core idea of *ZipIt!* so as to also apply to convolutional layers.

Concretely, consider corresponding convolutional layers $\mathcal{L}_1, \dots, \mathcal{L}_N$ in the models that we wish to merge. These layers operate through kernels (filters) $K_1, \dots, K_N$ of some shape of the form

$$\texttt{[out\_channels, in\_channels, filter\_size, filter\_size]} \tag{11}$$

that "sweep across" the input x, processing a local region at each step. The goal is to somehow combine these filters into a single filter K_{merged} that governs the corresponding convolutional layer of the merged model.

Recall that for a pair of corresponding linear layers $f_A = W_A x + b_A$ and $f_B = W_B x + b_B$, *ZipIt!* operates by looking at the pairwise correlations between the feature activations of the neurons that make up these layers. Pertinently, correlations are computed both *within* the same layer and *across* the two layers, the idea being that neurons corresponding to highly correlated features can be combined even through naïve means (e.g. linear interpolation) without losing too much information. We parallel this idea by viewing the filters $K_1, \dots, K_N$ as consisting of $n =$ `out_channels` subfilters (or "neurons") of shape `[in_channels, filter_size, filter_size]`, that is,

$$K_i = (k_i^1, \dots, k_i^n). \tag{12}$$

Analogously to *ZipIt!*, we then merge subfilters both *within* and *across* the parent filters $K_1, \dots, K_N$ according to pairwise "functional similarity".

4.3. How do we determine functional similarity?

In essence, functional similarity of the subfilters is determined similarly to the way it is determined for the neurons of linear layers under *ZipIt!*. Specifically, whereas *ZipIt!* determines the functional similarity between neurons i and j by looking at some empirical correlation measure between the feature activations of these neurons over a set of sample inputs, we parallel this idea by concatenating the filters $K_1, \dots, K_N$ into a single vector

$$\mathcal{K} = (k_1^1, \dots, k_1^n, \dots, k_N^1, \dots, k_N^n) := (k_1, \dots, k_{Nn}) \tag{13}$$

of Nn subfilters and looking at an empirical correlation measure between the outputs produced by feeding to $\mathcal{K}$ a large number of randomly generated data samples of shape

$$\texttt{[}N\texttt{out\_channels, in\_channels, filter\_size, filter\_size].}$$

Note that, as in *ZipIt!*, the concatenation is simply to allow these correlations to be computed both *within* and *across* the parent filters $K_1, \dots, K_N$, as opposed to just across these filters.

To be a bit more precise, each time a data sample is fed to $\mathcal{K}$, the output is a tensor of shape `[`N`out_channels,1]`. By arranging these outputs as the columns of a matrix, we obtain a tensor F of shape `[`N`out_channels, num_samples]`, the ith row of which can be thought of as providing a "holistic description" of the ith subfilter in the concatenated filter vector $\mathcal{K}$. Functional similarity between subfilters i and j is then determined by computing the *Pearson correlation coefficient* between rows i and j of this matrix. By doing this for each pair of rows i and j with i coming from the base model, we obtain $n \cdot Nn$ Pearson correlation coefficients that can be arranged naturally into a $n \times Nn$ *Pearson correlation matrix* P. In particular, this matrix has (i, j)-entry given by the Pearson correlation coefficient between rows i (subfilter i of the base model) and j (subfilter j of $\mathcal{K}$) of F.

4.4. How do we then produce the merged filter?

To produce the filter K_{merged} upon which the corresponding layer of the merged model shall operate, we draw inspiration from a few key observations:

1. Merging *pairs* of filters (as is done with the neurons of linear layers in *ZipIt!*) faces the problem that a given subfilter $k_i \in \mathcal{K}$ may be functionally similar to (in the sense of having high Pearson

correlation coefficient with) several other subfilters. In this case, we are failing to fully exploit redundancies in the models by only merging k_i with a single other subfilter that it is correlated with. A more natural approach is to take into account *all* subfilters k_i is correlated with, for instance by defining

$$k_1^{\text{merged}} := \sum_{j=1}^{Nn} P_{1,j} k_j, \quad \ldots \quad , k_n^{\text{merged}} := \sum_{j=1}^{Nn} P_{n,j} k_j \iff K_{\text{merged}} := P \cdot \mathcal{K} \tag{14}$$

In this example, we have produced a collection of n merged subfilters that receive influence from every other subfilter (both *within* and *across* the parent filters), with the strength of influence being governed by correlation. In particular, k_i^{merged} is heavily influenced by subfilters that are highly correlated with the base model subfilter k_i, and receives virtually no influence from subfilters that are not correlated with k_i. In other words, we can think of k_i^{merged} as being a blend of *all* subfilters (across all N models) that are correlated with k_i.

2. Suppose that two subfilters $k_i, k_j \in \mathcal{K}$ are functionally similar in the sense of having high Pearson correlation coefficient P_{ij}. This means that they are approximately linearly correlated, i.e. there is a nonzero number $m \in \mathbb{R}$ such that

$$k_j(x) \approx m k_i(x), \tag{15}$$

where $k_i(x)$ denotes the action of k_i on an input sample x, and similarly for $k_j(x)$. Notice that if $m \gg 1$, for instance, then this says that k_i is a lot like k_j but operates at a fraction of the scale. Merging according to $\frac{1}{2}(k_i + k_j)$ thus has the effect of producing something akin to $\frac{1}{2}k_j$, which (as a distorted version of the original subfilter k_j) is not likely to perform well. Evidently, we need to take this *difference in scale* into account. Precisely, prior to merging we should scale k_i up by m (or scale k_j down by $\frac{1}{m}$) to "level the playing field" and allow both subfilters to contribute meaningfully to the merge. This viewpoint becomes even more natural when we consider the case of $m < 0$ (i.e. the subfilters k_i and k_j are *negatively* correlated). In this case, interpolation according to $\frac{1}{2}(k_i + k_j)$ has the effect of producing a distorted (if $-1 < m < 0$) or inverted (if $m < -1$) version of k_i. Even worse, if $m = -1$ it produces a subfilter that (approximately) kills its inputs. By scaling k_i by m prior to merging, we prevent cancellation-based issues like these from occurring.

To take these observations into account, we encode the differences in scale via an $n \times Nn$ *slope matrix* S whose (i, j)-entry $S_{i,j}$ is given by the slope of the linear regression fitting the `num_samples` data points $(k_i(x), k_j(x))$. We then multiply this matrix entry-wise with the absolute value of the correlation matrix P to obtain a matrix C with (i, j)-entry given by $C_{i,j} = |P_{i,j}|S_{i,j}$, which we dub the *coefficient matrix.* The idea is that if $k_i, k_j \in \mathcal{K}$ are highly correlated, then $C_{i,j}$ is essentially just the scale factor that we wish to multiply by prior to merging. On the other hand, if k_i and k_j are not correlated, then $C_{i,j}$ is approximately zero. In other words, using the entries of C as merging coefficients provides us with a natural way of implementing the above observations simultaneously. With this in mind, we define the merged subfilters k_i^{merged} according to

$$k_1^{\text{merged}} := \frac{1}{Z_1} \sum_{j=1}^{Nn} C_{1,j} k_j, \quad \ldots \quad , k_n^{\text{merged}} := \frac{1}{Z_n} \sum_{j=1}^{Nn} C_{n,j} k_j, \tag{16}$$

where the $Z_i = \sum_{j=1}^{Nn} C_{i,j}$ are normalization constants that render each combination convex. Note that we can absorb these constants into the coefficient matrix C, at which point C becomes row-stochastic. Moreover, we note that since $k_i^{(\text{m})} = C_{i,\cdot} \cdot \mathcal{K}$, the merging process can be summarized more succinctly as

$$K_{\text{merged}} = \begin{bmatrix} k_1^{\text{merged}} \\ \vdots \\ k_n^{\text{merged}} \end{bmatrix} = C \cdot \mathcal{K}. \tag{17}$$

4.5. Sampling Distribution

Through iterative denoising, individual diffusion models learn to generate approximate samples from the distribution underlying their training datasets. A merged model, on the other hand, will sample according to some amalgamation of the sampling distributions of its constituents that depends on the merging process, and it is not at all clear what this sampling distribution looks like in general. A natural question to ask is thus: for models merged according to the *FeatureFusion* framework, what distribution does the merged model M_{merged} sample from, and can we express this in terms of the sampling distributions of the constituent models $M_1, \dots, M_N$?

To answer this question, we note that by (4) the sampling distribution of a diffusion model is uniquely determined by the score function it approximates, as this governs the distribution of the latent x_t at each timestep. In particular, if we can show that the score function $\nabla \log p^{(\mathrm{m})}$ of the merged model satisfies an identity of the form

$$\nabla \log p_t^{(\mathrm{m})}(x) = \nabla \log F(p_t^{(1)}(x), \dots, p_t^{(N)}(x)), \tag{18}$$

then it will follow that the sampling distribution $p_0^{(\mathrm{m})}$ of the merged model is given by

$$p_0^{(\mathrm{m})}(x) = Z^{-1} F(p_0^{(1)}(x), \dots, p_0^{(N)}(x)). \tag{19}$$

A reasonable strategy to this effect is to relate the output of the merged U-Net to those of its constituents. This is because the left-hand side of (18) is, by definition, approximately the output $s_{\theta_{\mathrm{m}}}(t,x)$ of the merged U-Net, and likewise the U-Net underlying model i satisfies $s_{\theta_i}(t,x) \approx \nabla \log p_t^{(i)}(x)$ for each $i = 1, \dots, N$. With this in mind, we recall from Section 4.4 that at each layer ℓ of merged U-Net, the output features f_ℓ^{merged} satisfy

$$f_\ell^{\text{merged}} = C_\ell f_\ell^{\text{cat}}, \tag{20}$$

where f_ℓ^{cat} denotes the concatenated vector $(f_\ell^1, \dots, f_\ell^N)$ of constituent features, and C_ℓ the coefficient matrix at layer ℓ. This tells us that we can relate the output of $s_{\theta_{\mathrm{m}}}(t,x)$ to those of the $s_{\theta_i}(t,x)$ via the product $U = \prod_\ell C_\ell$, and using this fact we obtain the following:

Proposition 4.5.1. *If $p_0^{(1)}, \dots, p_0^{(N)} : X \subset \mathbb{R}^d \to [0,1]$ denote the sampling distributions of the constituent models $M_1, \dots, M_N$, then the merged model M_{merged} (approximately) generates samples from*

$$p_0^{(\mathrm{m})}(x) = Z^{-1} \exp\left\{ \int_{\mathbb{R}^d} \log \prod_{j=1}^N \left(p_0^{(j)}(y)\right)^{K_j(x-y)} dy \right\}, \tag{21}$$

where the kernels $K_j : \mathbb{R}^d \to \mathbb{R}$ are defined by

$$K_j(x) = \mathcal{F}^{-1}\left[\frac{\xi^\top U_j \xi}{|\xi|^2}\right](x), \tag{22}$$

and U_j is the square minor of the matrix U corresponding to columns $(j-1)d+1$ through jd.

Proof. By the remarks of the preceding paragraph, we have that

$$\nabla \log p_t^{(\mathrm{m})}(x) \approx s_{\theta_{\mathrm{m}}}(t,x) = \sum_{j=1}^N U_j s_{\theta_j}(t,x) \approx \sum_{j=1}^N U_j \nabla \log p_t^{(j)}(x). \tag{23}$$

By taking the Fourier transform (component-wise) of both sides, we thus obtain that

$$\mathcal{F}\left[\log p_t^{(\mathrm{m})}\right](\xi) \cdot \xi \approx \sum_{j=1}^N \mathcal{F}\left[\log p_t^{(j)}\right](\xi) \cdot (U_j \xi). \tag{24}$$

By now taking the inner product of both sides of this identity with ξ, it follows that for all nonzero $\xi \in \mathbb{R}^d$ we have that

$$\mathcal{F}\left[\log p_t^{(\mathrm{m})}\right](\xi) \approx \sum_{j=1}^{N} \mathcal{F}\left[\log p_t^{(j)}\right](\xi) \cdot \frac{\xi^\top U_j \xi}{|\xi|^2}. \tag{25}$$

If we now define $K_j : \mathbb{R}^n \to \mathbb{R}$ by

$$K_j(x) := \mathcal{F}^{-1}\left[\frac{\xi^\top U_j \xi}{|\xi|^2}\right](x), \tag{26}$$

then it follows via Fourier inversion that

$$\log p_t^{(\mathrm{m})}(x) \approx \sum_{j=1}^{N} (\log p_t^{(j)} * K_j)(x) = \sum_{j=1}^{N} \int_{\mathbb{R}^d} K_j(x-y) \log p_t^{(j)}(y) dy = \int_{\mathbb{R}^d} \log \prod_{j=1}^{N} \left(p_t^{(j)}(y)\right)^{K_j(x-y)} dy \tag{27}$$

and hence

$$p_0^{(\mathrm{m})}(x) \approx Z^{-1} \exp\left\{\int_{\mathbb{R}^d} \log \prod_{j=1}^{N} \left(p_0^{(j)}(y)\right)^{K_j(x-y)} dy\right\}, \tag{28}$$

as claimed. □

Notice that if each U_j is $\frac{1}{n} I_{d\times d}$ (as it would be in the case of *Diffusion Soup*), then

$$K_j(x) = \frac{1}{n}\mathcal{F}^{-1}\left[\frac{\xi^\top \xi}{|\xi|^2}\right](x) = \frac{1}{n}\delta(x) \tag{29}$$

and hence the sampling distribution of the merged model reduces to

$$\frac{1}{Z}\exp\left\{\sum_{j=1}^{N} \frac{1}{n}\int_{\mathbb{R}^d} \delta(x-y) \log p_0^{(j)}(y) dy\right\} = \frac{1}{Z}\exp\left\{\sum_{j=1}^{N} \log\left(p_0^{(j)}(x)\right)^{1/n} dy\right\} = \frac{1}{Z}\prod_{j=1}^{N}\left(p_0^{(j)}(x)\right)^{1/n}, \tag{30}$$

which is consistent with the results of [3]. One can interpret (21) as an average of spatially weighted geometric means, with the weights $K_j(x-y)$ governing how much the density of model j at y influences the density of the merged model at x.

4.6. Through the Lens of Optimal Transport

Recall that if X is a Polish space and μ and ν are probability distributions on X, then a mapping $T : X \to X$ is said to be a *transport map* (between μ and ν) if T pushes μ forward to ν, that is, if $T_\# \mu = \nu$. More generally, a *transport plan* between μ and ν is a measure $\gamma(dx, dy)$ on the product space $X \times X$ with marginal distributions given by μ and ν. A transport map T is said to be *optimal* with respect to a given cost function $c : X \times X \to \mathbb{R}_{\geq 0}$ if it satisfies the optimization problem

$$T = \operatorname*{argmin}_{S : S_\# \mu = \nu} \int_X c(x, S(x)) \mu(dx). \tag{31}$$

Similarly, a transport plan γ is said to be optimal if it satisfies an analogous optimization problem for the quantity $\int_{X\times X} c(x,y)\gamma(dx,dy)$.

Another natural question to ask is whether the *FeatureFusion* framework can be formulated in some way through the lens of optimal transport theory. After all, given that the merging process is rooted in using

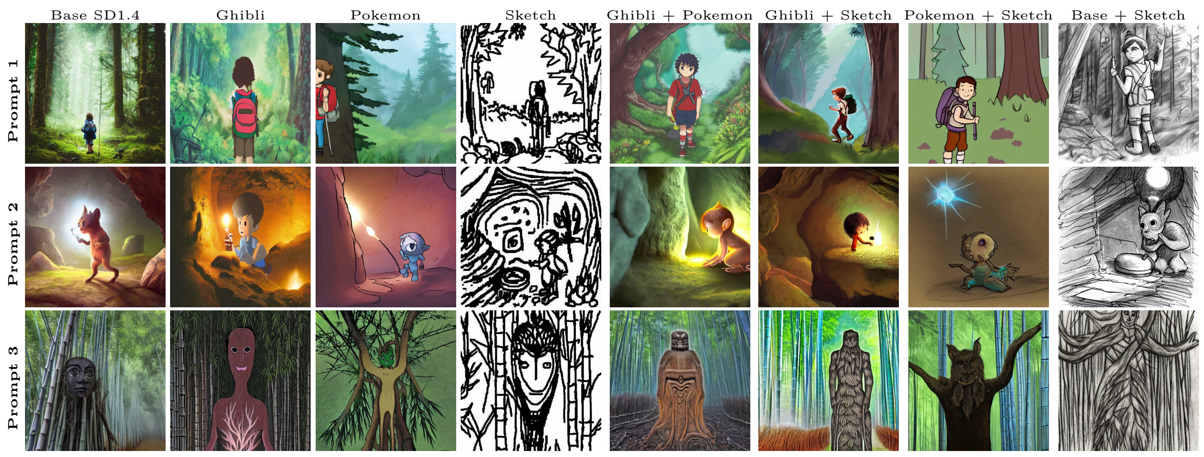

Figure 3: **Stylistic-base Qualitative Results.** Sample outputs of the base SD1.4 model, the stylistic specialist models, and various pairs of merged models. We utilize the following prompts:
Prompt 1: A young adventurer with a backpack standing at the edge of a magical forest.
Prompt 2: A small creature discovering a mysterious glowing artifact in a cave.
Prompt 3: An ancient tree spirit guardian with glowing eyes in a misty bamboo grove.

feature similarities to "prune redundancies" within and across the models, it is intuitive that the result should be "optimal" in some sense. One step in this direction is provided by the concluding remarks of Section 4.4, which imply that

Proposition 4.6.1. *Let μ denote the joint distribution of the features $f_\ell^1, \dots, f_\ell^N \in X \subset \mathbb{R}^d$ of the constituent models at layer ℓ, and let ν denote the distribution of the features f_ℓ^{merged} of the merged model at layer ℓ. Then the coefficient matrix C_ℓ at layer ℓ is a transport map between μ and ν in the sense that $(C_\ell)_\# \mu = \nu$.* □

Having established this, there is a sense in which C_ℓ can be framed as an *optimal* transport mapping to this effect:

Proposition 4.6.2. *The coefficient matrix C_ℓ at layer ℓ satisfies the optimal transport problem*

$$C_\ell = \underset{T:T_\#\mu=\nu}{\operatorname{argmin}} \int_{X^N} c(x, Tx)\mu(dx) = \underset{T:T_\#\mu=\nu}{\operatorname{argmin}} \mathbb{E}_{x\sim\mu}[c(x, Tx)] \tag{32}$$

with respect to the cost function

$$c : X^N \times X \to \mathbb{R}_{\geq 0}, \quad c(x, y) = |y - C_\ell x|^2.$$ □

Of course, one might object that this is "cheating" since we have *defined* the cost function c here so as to be trivially minimized by C_ℓ (in the sense that $c(x, C_\ell x) \equiv 0$). This is a fair point, and for this reason it is natural to wonder whether there is a more "natural" (or at least *non-circular*) cost function with respect to which C_ℓ is an optimal transport map between μ and ν. The answer is unfortunately no, since any reasonable choice for such a cost function should not depend on the layer ℓ. But if c were independent of ℓ then so too would be the solution to (32), and since C_ℓ is very much *not* independent of ℓ it cannot be the solution in such a case.

Any cost function with respect to which C_ℓ is optimal must therefore be "circular" at least in the sense of depending in some way on the layer ℓ. This, however, is not to say that there is no hope of achieving a stronger link to optimal transport than Proposition 4.6.2. One potential avenue for further work is whether there is a natural, layer-independent cost function c with respect to which C_ℓ is *approximately* optimal in the sense of satisfying

$$\left\| C_\ell - \operatorname*{argmin}_{T:T_\# \mu = \nu} \mathbb{E}_{x\sim\mu}[c(x, Tx)] \right\| \leq \epsilon_\ell \tag{33}$$

for some error threshold ϵ_ℓ dependent upon ℓ. Something else that may be interesting to explore is whether (under some fixed cost) ν approximates the Wasserstein barycenter of $\mu_1, \ldots, \mu_N$ with weights given by the entries of C_ℓ.

5. Experiments

5.1. Implementation Details

5.1.1. Datasets

We utilize Pick-a-Pic v1 [11], which is a publicly available dataset accessible via Hugging Face, as the base dataset for the object-based experiments. This dataset supplies highly diverse human-preference image-description pairs. Specialist data shards are then produced by searching for pre-defined key words in the image descriptions and assigning a score to each corresponding image category. The top 1000 images for each category are then split into training and test shards, used to finetune the base model, in turn producing a collection of specialist models. The categories chosen for this set of experiments include food, electronics, body parts, animals, text, vehicles, and clothing.

For the stylistic experiments, we instead opt for the FS-COCO dataset [5], which offers freehand sketches of common objects, and MSCOCO [12], an image corpus of common objects in context.

5.1.2. Benchmarking Metrics

To systematically evaluate the performance of each model, we employ widely used image-prompt metrics that assess both image quality and alignment with textual descriptions:

- **Image Reward [23].** A learned metric designed to assess image quality and relevance by modeling human aesthetic and semantic preferences. It provides a scalar score indicating how well an image aligns with human judgments of desirability.
- **CLIP Score [9].** A reference-free metric that quantifies image-text alignment by computing the cosine similarity between CLIP-encoded image and text representations. Higher scores indicate stronger semantic correspondence between the generated image and its associated prompt.

5.1.3. Models

As delineated above, to quantify the effectiveness of FeatureFusion, we train a collection of specialist diffusion models on both object-based and stylistic-based data shards. We finetuned the CompVis/stable-diffusion-v1-4 model [16] on the domain-specific data using a memory-optimized implementation. The process involved training only the UNet component while keeping the text encoder and VAE frozen to reduce computational requirements. Images were resized to 512×512 pixels and normalized before being encoded into the latent space. Training was conducted for 3 epochs on 1000 data samples using AdamW optimizer with a learning rate of 1×10^{-6} and a cosine learning rate schedule with warmup. To manage memory constraints, we implemented gradient accumulation with 2 steps, attention slicing, VAE slicing, and gradient checkpointing. The model was trained using MSE loss between predicted and actual noise in the standard diffusion denoising objective.

5.1.4. Computational Resources

The computational resources for this project included NVIDIA T4 and RTX 6000 GPUs, accessed through both the University of Toronto Computer Science department and Vector Institute GPU clusters. These

GPU resources were essential for handling the memory-intensive nature of diffusion model training, even with our optimized implementation that focused on reducing VRAM requirements.

5.2. FeatureFusion

5.2.1. Results

In this section, we present the main results from our proposed merging algorithm. Qualitative outputs and comparisons for the object-based specialists can be found in Figure 2. Similar outputs for the stylistic-based specialists are available in Figure 3, with each pair of constituent models being merged and presented. Finally, Figure **??** presents the quantitative benchmark performance for each of the object-based specialists, as well as the base SD1.4 model and the FeatureFusion merge of all seven specialists.

The object-based specialists show clear improvement in image quality compared to the base model (Figure 2). In particular, most adversarial artifacts are mitigated with the specialist models, and the prompt-image semantic alignment is visually far stronger. This is made particularly clear with the *vehicles* specialist. We also present quantified performance benchmarks for each specialist model on *all* object specialist tasks.

It is clear from Figure 3 that the stylistic base specialists visually fall into their niche artistic category. We additionally present pair-wise merged models between the different specialists. While merging of artistic styles can be difficult to parse and interpret, our merged models are able to effectively capture the style of the constituent models in a single image. This is most abundantly clear in the Ghibli + Pokemon merge, as well as the base+sketch model.

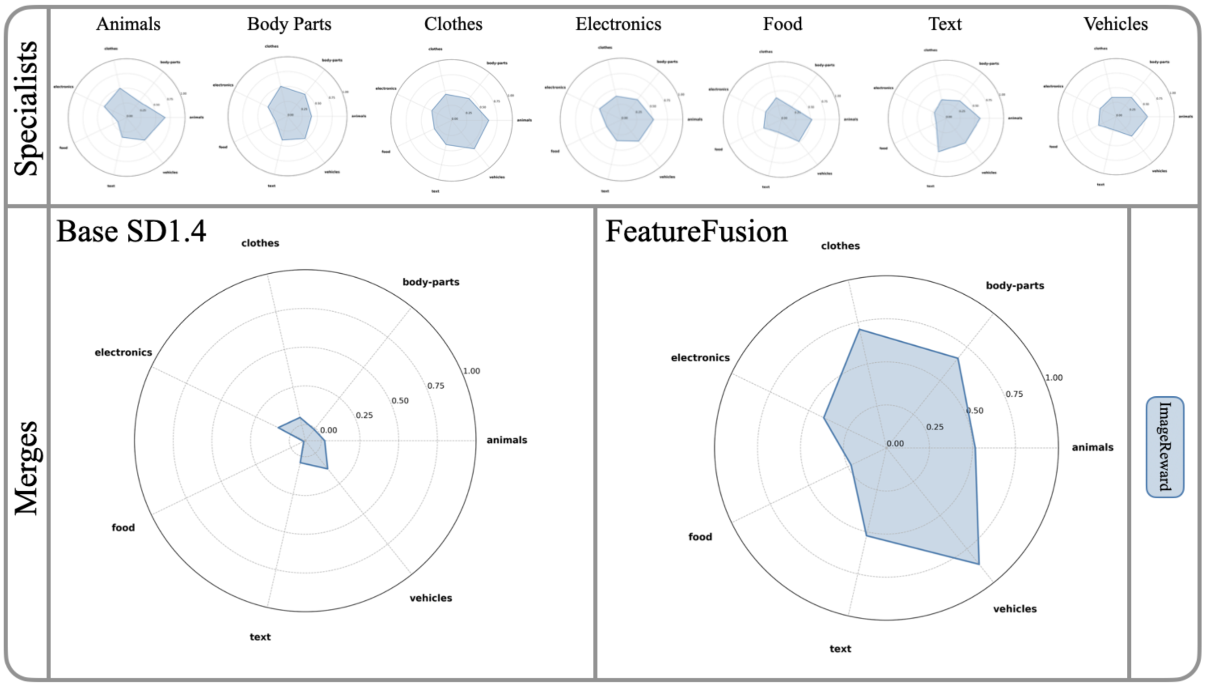

Figure 4: **Object-based Benchmarks.** Benchmark scores for the specialist models on each object-based task, as well as the performance of the base SD1.4 moel, and the *FeatureFusion* merge of the seven specialists. Each reported score is computed using the ImageReward score. See Section 5.1.2.

5.2.2. Computational Overhead

Let $\mathcal{M}$ denote the memory footprint (i.e. number of parameters) and $\mathcal{T}$ the forward-pass time complexity (i.e. number of diffusion steps) of the constituent models $M_1, \ldots, M_N$. Since the merged model M

produced by *FeatureFusion* has the same architecture as the constituents, the $O(\mathcal{M})$ memory and $O(\mathcal{T})$ inference complexities are preserved. Specifically, producing the coefficient matrices is a fixed $O(1)$ per-layer cost. Note that this outperforms ensembling methods [4, 25, 26], which have memory and forward-pass complexities of $O(n\mathcal{M})$ and $O(\mathcal{T})$, respectively. This also matches the optimal $O(\mathcal{M})$ and $O(\mathcal{T})$ complexities produced by the *Diffusion Soup* and *ZipIt!* frameworks [3, 21].

5.2.3. Unlearning

We've seen in Section 4.4 that, at each layer ℓ, the constituent model layers are merged through the coefficient matrix C. In particular, we have that

$$W_{\text{merged}} = C \begin{bmatrix} W_1 \\ \vdots \\ W_N \end{bmatrix} \quad \text{and} \quad K_{\text{merged}} = C \begin{bmatrix} K_1 \\ \vdots \\ K_N \end{bmatrix} \tag{34}$$

for linear and convolutional layers, respectively, where W is notation for a weight matrix and K a kernel. Equivalently, we can write

$$W_{\text{merged}} = \sum_{i=1}^{N} C_i W_i \quad \text{and} \quad K_{\text{merged}} = \sum_{i=1}^{N} C_i K_i \tag{35}$$

for suitable square minors C_i of C. It follows from this that if we wish to remove model j from the merge (i.e. *unlearn* the data shard covered by this model), then this is just as easy as merging. Specifically, in the case of linear layers, we simply need to re-define

$$\tilde{W}_{\text{merged}} := \tilde{C} \begin{bmatrix} W_1 \\ \vdots \\ W_{j-1} \\ W_{j+1} \\ \vdots \\ W_N \end{bmatrix} = \sum_{i=1, i\neq j}^{N} \tilde{C}_i W_i \tag{36}$$

where $\tilde{C}$ is simply the coefficient matrix C, but with the minor C_j removed and the rows re-normalized so as to maintain row-stochasticity. For convolutional layers the process is exactly the same (just replace all the W's with K's). Doing this layer-by-layer results in a new merged model from which the influence from model j has been removed.

6. Discussion

Our proposed *FeatureFusion* framework offers a new perspective on merging diffusion models by extending feature-correlation-based merging to convolutional layers. By constructing convex combinations of filters weighted by functional similarity and scale, *FeatureFusion* allows merging across different tasks while maintaining the computational efficiency of a single model.

Strengths of the framework include its theoretical grounding, its applicability to models trained on diverse tasks, and its minimal inference overhead. Additionally, we provided a formal description of the sampling distribution of the merged model and connected our method to optimal transport theory.

Limitations include the current reliance on identical architectures across models and sensitivity to the quality of sample inputs used for estimating feature correlations. Extending *FeatureFusion* to handle different architectures or improving its robustness to input data variations are promising directions for future work.

Open problems include:

- Extending the method to transformers and other architectures.
- Designing reference models that enable merging even when constituent models differ significantly.
- Exploring the properties of the merged sampling distributions in more detail.

By offering a more general approach to merging, we hope FeatureFusion paves the way for new methods of model composition, modular specialization, and efficient model reuse.

6.1. Reproducibility Statement

Our code is available in our open-sourced Github repository, which was directly used to produce the results presented in this paper. Additionally, each of our specialist models, as well as the cumulative merged model, have been made available on the Hugging Face Hub, and can be found here.

6.2. Beyond Same Architecture

While much of the prior work on model merging assumes that the constituent models share the same architecture (e.g., identical layer structures and dimensions), practical scenarios often involve models with both different initializations and different architectures. Extending merging methods to this more general case requires additional considerations.

One approach is to introduce a reference model that acts as a strict superset of the constituent architectures. The reference model must be flexible enough to accommodate the different structures of each input model. This can be achieved by:

- **Layer Generalization:** For layers with similar functions but different configurations (e.g., convolutional layers with different kernel sizes or transformer layers with varying head counts), design reference layers that generalize the operations. This might involve dynamically adjusting parameters like kernel size, number of channels, or attention heads.
- **Embedding and Projection:** When dimensionalities differ (e.g., different hidden sizes), use learned or fixed projection matrices to embed smaller models into the larger reference space. This allows all constituent models to be "lifted" into a common space where merging operations like averaging or alignment are possible.
- **Sparse Initialization:** Populate the reference model sparsely by placing weights from constituent models into corresponding regions, leaving uninitialized areas as zeros or learnable parameters. After merging, a fine-tuning phase can densify and optimize the resulting model.
- **Shared Representations:** Align similar modules across different architectures by learning a shared representation (e.g., matching feature maps or attention patterns) before merging weights. This alignment ensures the merged model respects the semantics learned by each constituent.

This framework introduces several challenges, such as ensuring that the reference model remains efficient, defining a meaningful matching between layers, and preserving performance from each original model. However, it offers a promising pathway toward merging models trained independently with different design choices, potentially enabling more powerful and generalized systems.

7. Conclusion

In this work, we introduced *FeatureFusion*, a new framework for merging diffusion models trained on different tasks without requiring additional training or inference overhead. Building on the idea of merging based on feature correlations, we extended this perspective to convolutional layers by treating filters as "big neurons" and constructing convex combinations weighted by functional similarity and scale.

Unlike existing methods that primarily interpolate weights or merge only pairwise based on correlation, *FeatureFusion* leverages all correlations across models, leading to a unified merged model that maintains the capabilities of the specialists. Through theoretical analysis, we characterized the sampling distribution of the merged model and established connections to optimal transport theory, offering a principled view of model merging.

Since our goal was to introduce a broader merging framework rather than outperform existing methods, we focused on theoretical development and qualitative validation over direct benchmark comparisons. Rather than aiming to outperform prior approaches such as Diffusion Soup or ZipIt! in a competitive benchmarking sense, our work provides a broader and more generalizable framework for integrating specialized models. We hope that *FeatureFusion* inspires further research into merging models with different architectures, tasks, or initializations under a unified theoretical lens.

Future directions include scaling to larger diffusion architectures, exploring the merging of transformer-based models, and extending the approach to non-identical architectures by constructing a common superset.

References

[1] Samuel K. Ainsworth, Jonathan Hayase, and Siddhartha Srinivasa. Git re-basin: Merging models modulo permutation symmetries, 2023.

[2] Brian D. O. Anderson. Reverse-time diffusion equation models. *Stochastic Processes and their Applications*, 12(3):313–326, 1982.

[3] Benjamin Biggs, Arjun Seshadri, Yang Zou, Achin Jain, Aditya Golatkar, Yusheng Xie, Alessandro Achille, Ashwin Swaminathan, and Stefano Soatto. Diffusion soup: Model merging for text-to-image diffusion models, 2024.

[4] Zixiang Chen, Yihe Deng, Yue Wu, Quanquan Gu, and Yuanzhi Li. Towards understanding mixture of experts in deep learning, 2022.

[5] Pinaki Nath Chowdhury, Aneeshan Sain, Ayan Kumar Bhunia, Tao Xiang, Yulia Gryaditskaya, and Yi-Zhe Song. Fs-coco: Towards understanding of freehand sketches of common objects in context, 2022.

[6] Prafulla Dhariwal and Alex Nichol. Diffusion models beat gans on image synthesis, 2021.

[7] Rahim Entezari, Hanie Sedghi, Olga Saukh, and Behnam Neyshabur. The role of permutation invariance in linear mode connectivity of neural networks, 2022.

[8] Timur Garipov, Pavel Izmailov, Dmitrii Podoprikhin, Dmitry Vetrov, and Andrew Gordon Wilson. Loss surfaces, mode connectivity, and fast ensembling of dnns, 2018.

[9] Jack Hessel, Ari Holtzman, Maxwell Forbes, Ronan Le Bras, and Yejin Choi. Clipscore: A reference-free evaluation metric for image captioning, 2022.

[10] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models, 2020.

[11] Yuval Kirstain, Adam Polyak, Uriel Singer, Shahbuland Matiana, Joe Penna, and Omer Levy. Pick-a-pic: An open dataset of user preferences for text-to-image generation, 2023.

[12] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *Computer Vision–ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part V 13*, pages 740–755. Springer, 2014.

[13] Michael Matena and Colin Raffel. Merging models with fisher-weighted averaging, 2022.

[14] Dustin Podell, Zion English, Kyle Lacey, Andreas Blattmann, Tim Dockhorn, Jonas Müller, Joe Penna, and Robin Rombach. Sdxl: Improving latent diffusion models for high-resolution image synthesis, 2023.

[15] Aditya Ramesh, Mikhail Pavlov, Gabriel Goh, Scott Gray, Chelsea Voss, Alec Radford, Mark Chen, and Ilya Sutskever. Zero-shot text-to-image generation, 2021.

[16] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 10684–10695, June 2022.

[17] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models, 2022.

[18] Tim Salimans and Jonathan Ho. Progressive distillation for fast sampling of diffusion models, 2022.

[19] Jiaming Song, Chenlin Meng, and Stefano Ermon. Denoising diffusion implicit models, 2022.

[20] Yang Song, Jascha Sohl-Dickstein, Diederik P. Kingma, Abhishek Kumar, Stefano Ermon, and Ben Poole. Score-based generative modeling through stochastic differential equations, 2021.

[21] George Stoica, Daniel Bolya, Jakob Bjorner, Pratik Ramesh, Taylor Hearn, and Judy Hoffman. Zipit! merging models from different tasks without training, 2024.

[22] Mitchell Wortsman, Gabriel Ilharco, Samir Yitzhak Gadre, Rebecca Roelofs, Raphael Gontijo-Lopes, Ari S. Morcos, Hongseok Namkoong, Ali Farhadi, Yair Carmon, Simon Kornblith, and Ludwig Schmidt. Model soups: averaging weights of multiple fine-tuned models improves accuracy without increasing inference time, 2022.

[23] Jiazheng Xu, Xiao Liu, Yuchen Wu, Yuxuan Tong, Qinkai Li, Ming Ding, Jie Tang, and Yuxiao Dong. Imagereward: Learning and evaluating human preferences for text-to-image generation, 2023.

[24] Zhengqi Xu, Ke Yuan, Huiqiong Wang, Yong Wang, Mingli Song, and Jie Song. Training-free pretrained model merging, 2024.

[25] Zeyue Xue, Guanglu Song, Qiushan Guo, Boxiao Liu, Zhuofan Zong, Yu Liu, and Ping Luo. Raphael: Text-to-image generation via large mixture of diffusion paths, 2024.

[26] Yanqi Zhou, Tao Lei, Hanxiao Liu, Nan Du, Yanping Huang, Vincent Zhao, Andrew Dai, Zhifeng Chen, Quoc Le, and James Laudon. Mixture-of-experts with expert choice routing, 2022.