UNIVERSIDADE FEDERAL DE PERNAMBUCO
CENTRO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

RODRIGO VITOR CASTRO ALVES DE MELLO

ELODIN: Naming Concepts in Embedding Spaces

Recife
2023

RODRIGO VITOR CASTRO ALVES DE MELLO

ELODIN: Naming Concepts in Embedding Spaces

Dissertação apresentada ao Programa de Pós-Graduação em CIência da Computação da Universidade Federal de Pernambuco, como requisito parcial para obtenção do título de mestre em CIência da Computação. Área de concentração: Inteligência Computacional

Orientador: Geber Lisboa Ramalho

Coorientador: Filipe Carlos de Albuquerque Calegário

Recife

2023

Catalogação na fonte
Bibliotecária Monick Raquel Silvestre da S. Portes, CRB4-1217

M527e Mello, Rodrigo Vitor Castro Alves de
ELODIN: naming concepts in embedding spaces / Rodrigo Vitor Castro Alves de Mello. – 2023.
46 f.: il., fig., tab.

Orientador: Geber Lisboa Ramalho.
Dissertação (Mestrado) – Universidade Federal de Pernambuco. CIn, Ciência da Computação, Recife, 2023.
Inclui referências.

1. Inteligência computacional. 2. Processamento de linguagem natural . 3. Deep learning. I. Ramalho, Geber Lisboa (orientador) II. Título.

006.31 CDD (23. ed.) UFPE - CCEN 2024-102

Rodrigo Vitor Castro Alves de Mello

“ELODIN: Naming Concepts in Embedding Spaces”

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de Pernambuco, como requisito parcial para a obtenção do título de Mestre em Ciência da Computação. Área de Concentração: Inteligência Computacional.

Aprovado em: 27 de setembro de 2023.

BANCA EXAMINADORA

_____

Prof. Dr. Giordano Ribeiro Eulálio Cabral
Centro de Informática/UFPE

_____

Prof. Dr. Andre Menezes Marques das Neves
Departamento de Design / UFPE

_____

Prof. Dr. Geber Lisboa Ramalho
Centro de Informática / UFPE
(**Orientador**)

Ao Criador de tudo e de todos.

AGRADECIMENTOS

Obrigado a meus pais, Laerte e Ceça, que sempre confiaram em mim e depositaram, num bebê, sua esperança. Obrigado a minha esposa, Raquel, que aguentou pacientemente e ad vertiu de forma doce diante dos exageros naturais da invenção. Obrigado a Christoph, Nat e Mahmoud da ModelMe UG, que foram mais do que companheiros de trabalho, cada um deles. Obrigado a Geber Ramalho e Filipe Calegário, meus orientadores, os quais contribuíram para esta dissertação e para o artigo original (pre-print) no qual ela é largamente baseada. Obrigado a você que não coube nesta margem estreita. Rebeca, Ricardo, Rodrigo Lopes, Rafael Cintra, André, Rafaela e muitos, muitos, muitos outros cujos nomes se perderam no tempo. Você, leitor, faz parte dessa história. Obrigado, também. Estamos juntos nessa.

Ah! Patrick Rothfuss, se você está lendo isto: Obrigado, em especial, por me ensinar a esperar. Digo isso pela história de "Wise Man 's Fear" e pela pessoa que você se tornou. And here’s one more thing for you to wait for, okay? I’m going to teach with the name of the Wind. I’d like you two to meet one day. It’s rude to give away someone else’s true name like that. Let’s go!

“Então Pilatos lhe perguntou: “Você não ouve a acusação que eles estão fazendo contra você?” Mas Jesus não lhe respondeu nenhuma palavra, de modo que o governador ficou muito impressionado.”

Mateus 27:13-14

ABSTRACT

Despite recent advancements, the field of text-to-image synthesis still suffers from the lack of fine-grained control. Using only text, it remains challenging to deal with issues such as concept coherence and concept cohesion. A method to enhance control by generating new words that can be reused throughout multiple images is proposed. Each new word, which I call “named concept”, can be mixed and matched freely with natural language, effectively expanding human vocabulary. Just as a painter combines pre-existing shades into personalized colors according to their needs, the proposed method enables combining e.g. “yellow” and “hawk” into a single word, that is, a single named concept. The new word, when present in subsequent text prompts, results in images that consistently contain the same yellow hawk. Unlike previous contributions, our method does not replicate visuals from input data. In some cases, it can generate visual concepts in a zero-shot manner, that is, without any visual input. A set of comparisons show our method to be a significant improvement over text prompts containing only natural language. Theoretical considerations on the foundations of Deep Learning are made throughout the text and Name Learning is proposed.

Keywords: artificial intelligence; embedding space; image generation; natural language processing; deep learning.

RESUMO

Apesar dos avanços recentes, o campo da síntese de imagens a partir de texto ainda sofre com a falta de controle no. Usando apenas texto, continua sendo desafiador lidar com questões como coerência de conceitos e coesão de conceitos. Eu proponho um método para melhorar o controle gerando novas palavras que podem ser reutilizadas em várias gerações. Cada nova palavra, que chamamos de “conceito nomeado”, pode ser misturada e combinada livremente com linguagem natural, expandindo o vocabulário humano. Assim como um pintor combina tons pré-existentes em cores personalizadas de acordo com suas necessidades, o método proposto permite combinar, por exemplo, “amarelo” e “falcão” em uma única palavra, isto é, um único conceito nomeado. A nova palavra, quando presente em prompts de texto subsequentes, resulta em imagens que contêm, consistentemente, o mesmo falcão amarelo. Diferentemente de propostas anteriores, esse método não replica visuais presentes em dados de entrada. Em alguns casos, pode gerar conceitos visuais de forma zero-shot, sem qualquer entrada de imagem. Um conjunto de comparações mostram a melhoria significativa da proposta sobre prompts de texto contendo apenas linguagem natural. Considerações teóricas sobre os fundamentos de Deep Learning são realizadas ao longo do texto e Name Learning é proposto.

Palavras-chave: inteligência artificial; embedding space; síntese de imagem; processamento de linguagem natural; deep learning.

CONTENTS

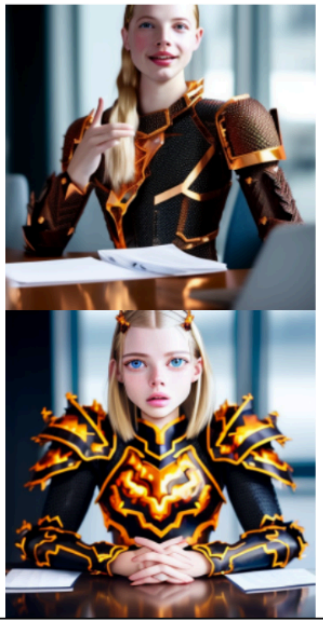
1 INTRODUCTION

The field of generative AI has grown significantly in the past years, experiencing one revolution after the other and achieving remarkable feats. In order to proceed even further beyond, there are still key challenges to be overcome, including limitations related to fine-grained control such as the ones tackled by recent editing works[4, 21, 39] and inversion works[12, 22, 32], which look into existing images (generated or otherwise) as a way of creating with precision.

In this context, I aim to provide a way of persisting precise visual concepts in a zero-shot fashion, that is, without relying on pre-existing images. The solution involves searching for certain vectors in the embedding space of encoders, generating special “named concepts" that expand natural language vocabulary much like new words. I call this process *concept naming* and the method, *Elodin*[1] Experiments have found Elodin to be successful in that goal. To support my claims, I provide a clear set of side-by-side qualitative comparisons, as well as quantitative measures based on face similarity.

[1] Elodin is a fictional professor from a popular fantasy book (The Name of The Wind) who knows the hidden names of things

Table 1: images generated with namecons. <Lucy >, <yellow bird >, etc. represent “new words” that are generated with AI. Top row was made with the fist Stable Diffusion release (SD 1.4). Bottom row was made with a Dreamshaper model available at (civitai.com). Code and all parameters are the same in each column, only the checkpoint le was changed

<Lucy >wearing a <galaxy tunic >at the beach	<Lucy >in a <fire armor >at a business meeting	<Lucy >wearing a dress made of <cloth >for a party	<Lucy >wearing red petting a <yellow bird >
			

Source: the author (2023)

2 PROBLEM CHARACTERIZATION

Prompting an image synthesis model with natural language descriptions has a number of advantages over the traditional method of manually defining each pixel's colors (by e.g. brush strokes), as the popularity of recent initiatives such as Stable Diffusion[31] can attest. However, it is not without some disadvantages; notably, the lack of fine-grained control.

I focus on two issues that exemplify this lack of control. The first one, which I refer to as the *concept coherence* issue, speaks to the difficulty of expressing the same visual concept over multiple runs. For instance, an image for a person's face can be easily crafted by an SD user by providing a prompt such as “a face of a person". In some cases, however, there is a need to produce more images of that “same person" in different settings (such as at the beach, in a business meeting, or, simply, in another pose). Unfortunately, in these cases, it is challenging to guarantee that the faces in the newly generated images will look alike, even using a long and precise input prompt, as illustrated in Table 2. In this example, although every generated face meets the description, they do not appear as the same person, even using the same seed for each column.

More generally, one may wish to maintain the coherence of any particular visual concept, be it the appearance of a person, object, scene, texture, palette or artistic style, between multiple runs from the same natural language prompt. It is not obvious how to accomplish that, since there is no precise natural language description for most visual concepts (e.g. how to precisely describe a specific face, unless it is the face of a famous person who is well represented in the dataset?).

Table 2 Generations from an extended and descriptive prompt (first row: “the face of a middle-aged brunette woman with blue eyes and a thin nose at the beach“ and second row “the face of a middle-aged brunette woman with blue eyes and a thin nose at a business meeting”), with commonly used negative prompt “bad artist, bad perspective” and the same random seed in each column. SD 1.4

Source: the author (2023)

This issue affects visual storytelling applications[18], such as video clips, games, graphic novels, etc. Such a need is not restricted to storytelling, however; it may impact applications such as product design and publicity.

The other issue, which I call “*concept cohesion*", arises when a certain visual concept interacts with others in an unintended way. The most obvious example is color: a prompt such as “a yellow hawk amidst white flowers" often results in some yellow being applied to the flowers, or in a hawk that is not particularly yellow (Tables 3).

Table 3 prompt “a yellow hawk amidst white owers”. SD 1.4 model.

Source: the author

Table 4 prompt “a warrior wears armor made of fire and lava on a frozen mountain peak”. SD 1.4 model.

Source: the author

Even though the focus of this report revolves around visual concepts, those issues may in principle be generalized to other kinds of concepts. For example, it is also hard to keep concept coherence in writing style throughout text-to-text generation [14].

3 RELATED WORKS

3.1 Overview

Many solutions have been proposed to generally enhance control in text-to-image models. ControlNet[42], for instance, achieves shape control via edge maps, line drawings, segmentation maps, etc., while Latent Guidance[38] uses sketches to guide the process. Other works extract style, palette, etc. from one image and apply them to another[16, 1]. There are also popular tools (OpenAI DALL-E 2[2], DreamStudio[3]}, AUTOMATIC1111 webui[2]) with features such as image-to-image and inpainting, which help provide the model with more information than would be possible with just a simple text prompt.

Focusing on the problem of concept coherence, there is a class of notable methods stemming from PerVL[5] such as Textual inversion[12] and Dreambooth[32]. Through the process of *inversion*, these methods enable the reproduction of the same visual concept across many runs, as long as this concept can be represented by input images. Those methods often achieve high coherence between multiple representations of the same object. A key limitation among these methods is the need to have the concept already expressed in visual form, often in multiple images. For instance, to generate the same face across multiple runs with Dreambooth, it is necessary to have multiple pictures of that same face from different angles as input data.

For some of these methods, there are also other limitations beyond the need for image input, such as difficulty in combining multiple inverted visual concepts in the same run and long training times. Some of these other limitations are explored in more recent works[22, 13]. A concurrent work to this one that focuses on face coherence and strongly overcomes these limitations is[41].

The other issue, cohesion, is also tackled by inversion techniques, albeit in a cumbersome way. In the “a yellow hawk amidst white flowers'' example, one could,

[2]https://labs.openai.com/
[3] https://dreamstudio.ai/

first, generate some pictures of a yellow hawk, which would then be used as input to the inversion method[18]. The result of the inversion would, at a later time, be combined with the white flowers. For some situations, the initial generated objects might be different from one another (such as different yellow flowers), thus hindering the inversion process.

There is a concurrent work which is similar to this one[40]. It presents the ConES (Concept Embedding Search) technique, which was developed independently from Elodin[24], upon which this dissertation rests.

Outside of computer science, acknowledgements are in order to Kevin Scharp and his work in the foundations of logic. In hindsight, some of the inspirations present here resemble the field of Conceptual Engineering. I did not, however, apply any conceptual engineering technique in a direct manner, nor did I apply any book other than Scharp's Replacing Truth[35]. These works in philosophy are deeply related to the present discussion. That said, I vehemently disagree with replacing concepts such as truth and justice; rather, I try to use truth in the usual sense often and try to employ engineered concepts when a need arises.

3.2 PerVL, TI and Dreambooth

Perhaps the closest related works to this are “Personalized Vision and Language" (PerVL), Textual Inversion and Dreambooth. PerVL pioneered a new setup for learning, in which a model's vocabulary V is expanded to V' = V U C where C is a new set of concepts C = {c1, c2, ..., ck}. Being a general training setup, PerVL cannot be evaluated directly (in a similar manner to Name Learning). Therefore, PerVL was evaluated through PALAVRA, which is a particular algorithm that follows the PerVL setup. In that sense, PerVL is equivalent to Name Learning and PALAVRA is equivalent to Elodin. Advantages of the PerVL setup include fine-grained personalization and lightweight processing requirements, while its limitations make it ill-suited for dealing with large datasets.

PALAVRA works by learning an inversion mapping from a set of points in the image space of CLIP back to the embedding space. This inversion mapping is used to

iteratively adjust an embedding through backpropagation until it matches a given set of images. One of the main innovations of Textual Inversion is to replace this inversion mapping with the usual training cycle of an LDM model [30]. The new method, then, produces personalized embeddings that can be readily applied to well-known neural network architectures and that express fine visual detail. At inference time, it is enough to replace one of the embeddings that would be generated from the prompt with the new embedding made with textual inversion, as detailed in section 7.2, “Using namecons" (the usage of namecons at inference time is similar to the usage of textual inversion embeddings)

To the basis established by PerVL and Textual Inversion, Dreambooth adds numerous enhancements. Of those, two are most related to this work: rare-token identifiers and class-specific preservation loss. Rare-token identifiers are words which carry little meaning in daily usage, such as the 3-character apparently random string ``sks" (which is not directly mentioned in the dreambooth paper). These tokens are used as placeholders for the embedding, as in (“sks amidst white flowers"). This is so the program can replace whatever embedding would naturally result from “sks" with the embedding created by dreambooth. Class-specific preservation loss relates to model updates alongside with embedding creation. In PALAVRA and Textual Inversion, the base neural network is kept frozen (only the embedding receives updates from backpropagation). With the introduction of class-specific preservation loss, the base neural network is also updated, thus resulting in even more precise, clear and flexible final results

In the Elodin algorithm (which relates to Name Learning in the same way PALAVRA relates to PerVL), no class-specific preservation loss is employed. There is also no use of rare-token identifiers. The resulting embeddings, which are specifically called "named concepts" or "namecons" are compatible with any CLIP-based system (such as any Stable Diffusion model derived from SD 1.5) without any further adaptation. Namecons carry specific, precise and flexible visual concepts in a file that's just a few kilobytes long. They can also be combined with one another freely, as well as with word from natural language. In all that, name learning differs from previous methods. Notably, the concepts which are stored in namecons are not pre-existing, but are created with A.I. For example, with Textual Inversion, one might create an embedding

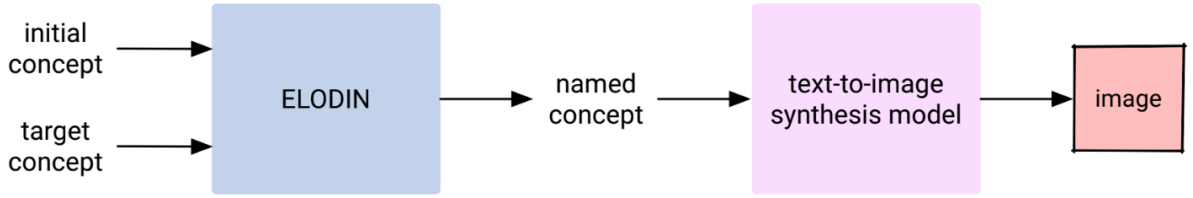
containing the appearance of a real-world coffee mug. With name learning, one might create a namecon containing the appearance of an AI-imagined coffee mug. The fact that we name concepts which were previously unknown is the reason for the name “named concept".

4 PROPOSAL

Considering the previous challenges, as well as both the limitations and accomplishments of current techniques, I propose a process that is centered on the idea of assigning a custom keyword to a particular concept, even if that concept cannot be easily expressed in natural language. An analogy would be a person's proper noun: even though someone's facial features cannot be easily put into words, their name can be used instead (indeed, when prompted with famous people's names, some models can produce their face as resulting image). Therefore, for example, one may name a generated person <Lucy>

As another example, to produce a yellow hawk amidst white flowers, one might generate a visual concept about a specific yellow hawk (calling it <my_hawk>, for instance) and, then, use that word as part of a prompt (“<my_hawk> amidst white flowers''). All the resulting images would feature that particular yellow hawk and, since there is no direct mention to its color in the prompt, the flowers would be indeed white. In other words, the prompt does not contain the word “yellow'', so it is impossible for the model to be confused. The yellow color, in this case, would be inherent to the bird, much as it is inherent to a banana.

Figure 1 Summarized proposal



Source: the author (2023)

The whole process is roughly illustrated in Figure 1 showing only the inputs and outputs. More details will be added in the next chapters. The role of the *initial concept*} (e.g. ``bird'', ``woman'', ``armor'', or ``cloth'') is to provide a starting point for the *concept naming* process, represented in the figure by the Elodin algorithm (this particular concept naming algorithm will be discussed in section 6, "Method"). The *target concept* is the specialization of the initial concept one wishes to aim for. That

is, the target concept adds qualities to the initial concept. The initial concept can be understood as a course descriptor to which more qualities are added. Considering previous examples, the respective target concepts could be, for instance: “a yellow hawk", “beautiful blonde woman", “armor made of fire and lava", or “a very smooth celadon dress".

When the concept naming process converges, it generates a new concept that cannot be expressed by natural language. For example, a specific virtual bird which is yellow and looks like a hawk, or a specific virtual female face which is blonde and beautiful, a specific virtual armor with fiery motifs, or a specific virtual texture that resembles smooth cloths such as silk or linen and is always grayish-blue. Please note that, while these examples describe approximations of each concept, they do not describe the concept itself. Each reader to this paragraph will imagine different virtual birds. All generated images, however, will present the exact same one.

After the concept is generated, it is stored in an embedding file (much like in Textual Inversion). To complete the creation, it is necessary to associate the embedding file to a string, so it can be referred to in the text prompt and combined with words from natural language. This allows the user to type “<my_hawk> amidst white flowers" and have the system recognize that <my_hawk> refers to that embedding file. I call the final product a *named concept* or simply *namecon*. Revisiting our previous examples, one could have created, respectively, the following namecons: <my_hawk>, <Lucy>, <fiery_armor>, or <my_cloth>

Namecons can also be freely used together in the same prompt. (“<Lucy> wearing a dress made of <cloth> while petting <my_hawk>").

Differently from previous techniques, the goal of concept naming is not to invert some images into their corresponding inner representations (embeddings), but, rather, to compute representations for new concepts. Therefore, namecons can be created without images, relying only on natural language prompts. In other words, it is possible to generate visual concepts from textual descriptions alone.

It is also possible to input a visual description to the Elodin method. For example, one may input a picture of a real person as the target concept instead of the text “beautiful blonde woman". In that case, the concept that is created when the algorithm converges resembles some broad characteristics of the person portrayed in the picture (such as the same ethnicity), while clearly generating a different face. The reader may be questioning if there is any difference between inputting a picture or the name of the same famous person (e.g. inputting the text ”Abraham Lincoln" or a picture of him). The answer is that the final generated faces look more natural if the target concept is a picture of a face, instead of a name. All faces reported in this document as being created with the new method are generated from prompts containing namecons that were created using pictures as visual target concepts.

In sum, the proposal is Elodin, an algorithm to generate concepts with A.I. and store them in embedding files. Elodin was implemented in the context of text-to-image generation. This implementation can take text or pictures as input and output embeddings, which, then, can be used to generate images through the usual process introduced by PerVL[5] and popularized by Textual Inversion[12].

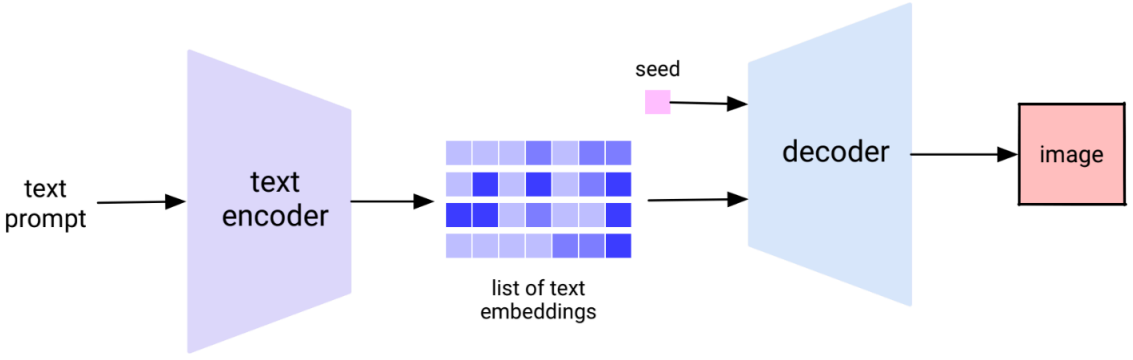
5 FUNDAMENTALS

To understand the details of the Elodin method for naming, it is helpful to first consider the following:

5.1 Text-to-image pipelines

Recent large-scale text-to-image models, such as DALL-E 2 [28], Imagen [33], Latent Diffusion [30] and StyleGAN-T [34] are *not* comprised of a single network that translates natural text to image. Instead, in all of those different architectures, the natural language prompt is first input to a language model, such as CLIP[25], BERT [7], T5[27], etc. Those language models then generate an embedding that is used as conditioning for the actual image generation process. Therefore, it is not precise to refer to these mechanisms as “a text-to-image model", but rather as “a text-to-image pipeline", since they are not comprised of a single neural network (the terms ``model" and “neural network" are used equivalently in Deep Learning, so a pipeline is comprised of more than one neural network). This distinction becomes very relevant as namecons are an input to the second network, that is, the image generation part of the pipeline, instead of being an input to the first network (the language model) at the start of the pipeline.

FIgure 2 Generic text-to-image pipeline



Source: the author (2023)

In general terms, as illustrated in Figure 2, such pipelines can be understood as starting with a *text encoder* component, which translates raw text into a kind of inner representation called *embeddings*. In usual implementations, each prompt generates a fixed number of embeddings, no matter the prompt's length. Longer prompts are

truncated to this length and shorter prompts are automatically filled with “end of sentence'' symbols until they generate that fixed amount of embeddings. Therefore, it is possible to simplify a text encoder as a black-box that takes in text and always outputs a known amount of embeddings. The text encoder is a neural network.

Each embedding is usually implemented as a tensor. For this discussion, please consider them as unidimensional sequences of floating point numbers. Therefore, since a text encoder outputs a known number of embeddings and each embedding is a sequence of numbers of known length, it is possible to view a text encoder as a black-box that takes in text and outputs a 2-dimensional matrix of floating point numbers. Each row, for this discussion, corresponds to an embedding. For those of a technical background, the amount of rows is the context length and the amount of columns is the embedding dimension.

If a user inputs a short text, such as just “bird'', the first row of the matrix generated by the text encoder will roughly correspond to that word. The other rows (since it is always a fixed size) will roughly correspond to the emptiness between ``bird'' and the maximum amount of words that the encoder accepts. In technical terms, row 0 is heavily influenced by the [SOS] token, while the row 1 is heavily influenced by the “bird'' token; all other embeddings correspond to the [EOS] token

Please take note that, while some embeddings correspond to some input words, it is not a clear correspondence. Other rows will also be influenced by the fact that there is a “bird'' in the sentence, even if they relate to a different part of the text. Text encoders are built this way in order to deal with textual context (knowing whether to use “king'' or `”ueen'' in a sentence, for example). It is, thus, more productive for the present discussion to consider the entire output matrix as relating to the entire input text.

After the matrix is calculated by the text encoder, it is used as input to the image *decoder*. The image decoder is also a neural network. It is important to point out that no other information is communicated between those modules; in fact, many successful implementations such as Imagen[33]and DeepFloyd IF [paper to be released] start from a pre-made ("frozen") text encoder which was not trained on

images and, then, train an image decoder using only embeddings (generated from the text encoder), instead of actual strings of text. Therefore, the image decoder is quite independent from the text itself; rather, it depends on the text encoder and its embeddings. Similarly, the text encoder is often fully independent of the image decoder (the image decoder is often created at least a year later). Finetunings of the same base image decoder can be swapped one for another in the pipeline without any adaptation, as exemplified in the first picture of this document and practiced by model sharing communities. Different finetunings of the same base image decoder can create images in different styles, such as photorealistc or cartoonish, from the same set of embeddings (that is, the same matrix of floating point numbers).

In summary, “text-to-image models" refers, in practice, not to one model, but rather to multiple models in a pipeline. These neural networks are often not trained together; rather, they are often trained by different teams of researchers years apart from each other (although each team needs to have access to the work of the previous ones). This endows model pipelines with much independence among their components, as in the example of swapping many image decoders without having to change the text encoder. For this dissertation, I focus on the data that is transmitted between these components, which is usually represented on code as a 2-dimensional matrix of floating point numbers. Each row of this matrix corresponds to an embedding. I will, therefore, refer to the whole matrix as simply “embeddings", in the plural.

In text-to-image pipelines, a raw string of text is transformed into embeddings by the text encoder. The proposal presented in the previous chapter is about changing the embeddings before they are sent to the image decoder. Suppose a user inputs “a yellow hawk amidst white flowers" to the text encoder. Also, suppose the user has access to a file containing an ai-generated embedding for <my_hawk>. Then, it is possible to replace the embeddings corresponding to “a yellow hawk" with the previously generated <my_hawk>. After that, the new embeddings can be sent into the image decoder. The final result is an image of the user's hawk amidst white flowers. Previous works[5, 12]} have employed similar mechanisms to apply embeddings to the image generation process (more details in chapter 6, "Method"); the Elodin algorithm is a novel way of creating the embeddings themselves.

5.2 Embedding space: latent vocabulary and latent lexicon

Unconditional image generators [15, 26, 19, 10, 8] are the fundamental kind of neural-network-based image synthesis software. This type takes in noise samples (usually Gaussian) as input and map them to output images, mimicking the probability distribution of the images in the training dataset. The entire process (whether a single neural network or a pipeline) can be seen a black box which map each point in a random distribution to a specific generated image in a deterministic way. The stochastic nature (pseudo-randomness) comes from choosing the initial point in the random distribution through a pseudo-random number generator. In the user interface, this number is often referred to as “generation seed" or simply “seed". The same seed always results in the same image (keeping other parameters the same). For this kind of generation, a text encoder is not necessary. Unconditional image generators do not need to be implemented as pipelines.

Conditional image generators [3, 10, 8, 31, 29] are the most common type. Stable Diffusion is a widely known example. Those start from the basis of an unconditional generator and adapt it to take other inputs (in the case of Stable Diffusion, mainly text) into consideration. In technical terms, the image generation network is conditioned with embeddings originating from other modalities, such as as one-hot class labels, text strings, or pre-existing images. For text, first the string is put through a pre-trained text encoder to generate embeddings. Then, the embeddings are used as conditional input to train the image decoder from scratch. This description is very simplified, see the original work on Latent Diffusion[30] for details. For conditional image generation from natural language, it is more practical to implement a model pipeline (there have been no successful single-network implementations that I know of). Conditional image generation from other modalities, such as class labels, have been successfully implemented [3] as single generator networks.

Therefore, image decoders for conditional image generation do not interact with text, but, rather, with the embeddings that are generated by the text encoder. Each single embedding can be construed as a mathematical vector (implemented as an array of floating point numbers) in a vector space. This vector space is often called “embedding space". For the purposes of this explanation, it is relevant to introduce a

distinction between two implicit meanings of the expression “embedding space". These are: “latent vocabulary" and “latent lexicon".

Latent vocabulary refers to the embeddings that the text encoder can generate. In the most common implementation of CLIP, for instance, the text encoder always outputs 77 embeddings; there is also a limit for the length of the input text. In that example, the latent vocabulary would be all sets of 77 embeddings that can be output from a given CLIP text encoder. This is the latent vocabulary of a text encoder.

Latent lexicon refers to the embeddings that can never be output by a given encoder but that can be used as input to some decoder trained on the latent vocabulary. For example, in most Stable Diffusion versions, the text encoder is a pre-trained encoder from CLIP. Many different image decoders can be trained on the embeddings of that pre-trained encoder (such as different Stable Diffusion checkpoints, finetunings, versions, and other non-Stable DIffusion decoders), either from the same image dataset or from different image datasets. It has been shown in PerVL[5] that it is possible to use an embedding that was not created by the text encoder as input to one of those image decoders in a well-behaved way. The collection of embeddings that can be used as input to decoders which were trained on the vocabulary of a given encoder (and not on additional embeddings of any other neural network) is the latent lexicon of that specific encoder.

Given that embeddings can be construed as mathematical vectors in a vector space, it is also natural to think of the latent lexicon as the interpretable portion of that space, which contains many more points than the points that the encoder is able to compute.

In summary, the latent lexicon of an encoder is the collection of all “words" in the language to which the encoder translates its input. In the context of text-to-image generation, the text encoder translates from natural language (usually English) into the “language" of embeddings. This text is, then, passed on to the image decoder, which translates the embeddings into the “language" of images. In each step (English, embeddings, image), the meaning is kept the same: “a yellow hawk amidst white flowers", although some details may be lost in translation.

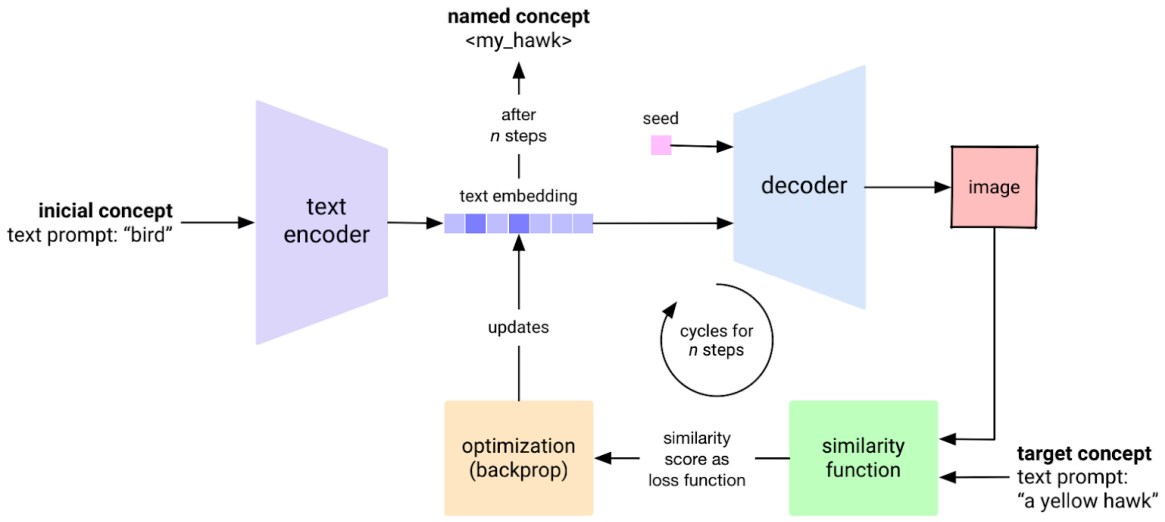
The latent vocabulary, then, is the “words" in the encoder's language that the encoder can effectively generate. Similarly, I cannot speak every word of the English language; the ones I know form my English vocabulary.

The Elodin[24] algorithm can be thought of as a search within the latent lexicon. Namecons are embeddings found through that search process and labeled <my_hawk>) for future use. “Naming" is the process of searching and saving a namecon. Given that naming does not involve training nor inference, it can be understood as a third action from a nascent field which does not create nor update neural networks. That field would be a different kind of Representation Learning (the only successful kind of representation learning, currently, is Deep Learning). I propose this new field be called Name Learning.

6 METHOD

In this chapter, I dive into the technical details of the proposal briefly discussed in chapter 4, "Proposal".

Figure 3. The Elodin method



Source: the author (2023)

6.1 Creating namecons

To generate a namecon, begin by inputting to an encoder a prompt corresponding to the initial concept (as defined in chapter 4, "Proposal"), such as "bird", "armor", or "woman". Figure 3 details the process, taking “bird' as the initial concept. The encoder generates embeddings, which are, then, fed to the decoder to produce an image. After image generation, compute the similarity between the output image and the “target concept" (as defined in chapter 4, "Proposal"), such as “a yellow hawk". Then, use the similarity score as a loss function to optimize the embeddings through backpropagation. Repeat the process from embeddings to image, then to similarity score, then to backpropagation (which updates the embeddings) until the image reflects the target concept instead of the initial concept. Since, at each iteration, the text encoder is not used (it is only used once for the first iteration), the resulting image will have been produced by the updated embeddings. Therefore, the updated embeddings will carry the information necessary to make an image resembling the

target concept and, thus, will be ready to be used to generate other images reflecting that concept. In other words, the final embeddings will no longer refer to "bird"; instead, they will refer to "a yellow hawk". Save these embeddings in a file in the usual manner and associate with them a custom name, such as "my_hawk". The pair made of the embeddings file and the custom name is the namecon, referred to as <my_hawk>. This is the description of the Elodin algorithm.

The explanation above is approximated. The main details that need to be added are, first, that not all embeddings are updated. Rather, from the initial collection of embeddings produced by the text encoder, only some are selected to be updated. The second main detail is that the output image is produced from a different random seed each time to avoid overfitting. These seeds can all be produced by a regular pseudo-random generator, so mentions to "the seed" relate to the initial seed to that generator. Third, it was mentioned in chapter 4, "Proposal" that the target concept could be an image instead of a textual description. This requires only the adaptation of the similarity function. Finally, one may ask why is the namecon different from the embeddings that would result from simply inputting the target concept to the encoder. These topics will be explored in the following subsections.

6.1.1 *Which embeddings are updated*

Taking as example the initial concept "bird", the actual sentence that was input to the text encoder in the experiments was similar to "a photo of a bird". The added words help provide context to generate the image at each iteration. Please recall (as discussed in chapter 5, "Fundamentals") that the output of the text encoder always contains the same amount of embeddings (at least, for current implementations). Of all embeddings, only the ones that correspond to the initial concept are updated at each iteration.

6.1.2 *Seed*

The image decoder takes as input a random seed (since it's based on an unconditional image generator, as discussed in chapter 5, "Fundamentals"). For each iteration of the Elodin loop, the random seed is different. This is to prevent overfitting

to a specific image layout (since the seed controls aspects such as layout, small details, etc.). It is possible to provide only one seed at the start of the whole process to a random number generator that will produce subsequent seeds, or simply to increment the initial seed by 1 at each iteration. Another way to avoid overfitting is to produce a batch of images per iteration (each from its own seed) and take the average of each similarity score.

6.1.3 *Images as target concepts and other loss functions*

The similarity function employed in the experiments was the standard cosine similarity based, again, on a CLIP model. To employ images as target concepts, one simply inputs the image to the CLIP cosine similarity function. For clarity, by "standard cosine similarity" I mean taking the generated image at each iteration and using it as input to a CLIP network. Then, taking the target concept (either a text string or another image) and also using it as input to the same CLIP network. This will result in two standard CLIP embeddings, which are (as discussed in chapter 5, "Fundamentals") vectors. By taking the cosine between these vectors, one arrives at a single scalar that is the similarity score. This similarity score is used as loss function to the backpropagation.

All images in this report that feature faces made with Elodin were created from the same <Lucy> namecon. That namecon was created using an image of a person as the target concept, not a text string. This results in a more natural-looking final result. The image used as a target concept does not need to depict an actual human; a face generated with another network such as Stylegan 2[20] suffices.

Other similarity functions could, in principle, be used. For example, if one can implement a similarity function between shades of color, it would be straightforward to generate namecons for specific color shades or specific color palettes.

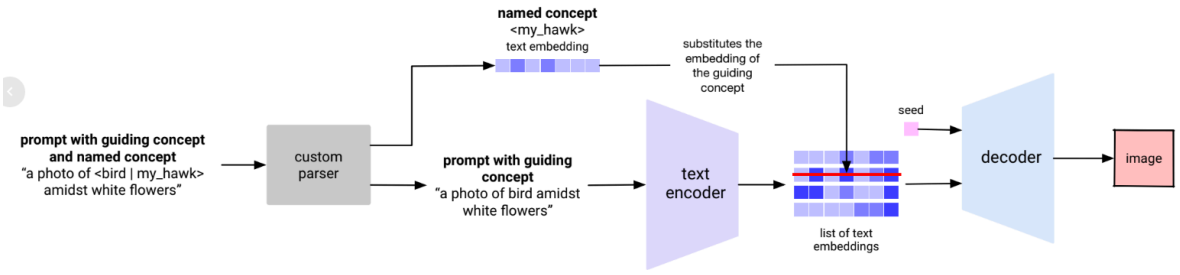
6.1.4 *Why namecons are different from regular embeddings*

By "regular embeddings" I mean the embeddings that result from simply inputting the target concept into the encoder. For example, just typing in "a yellow hawk amidst

white flowers" to the encoder generates a set of embeddings. Using these embeddings results in images which are very different from the images resulting from using <my_hawk> amidst white flowers" (the images resulting from a prompting containing this namecon depict very yellow hawks and contain much fewer yellow flowers, if any). This may come as a surprise to the reader, since <my_hawk> was created to iteratively approximate the concept of a yellow hawk. Though I do not have a final answer, this is my attempt at an explanation: due to the modality gap[23], embeddings generated in this regular way cannot accurately represent visual concepts even in a shared embedding space. I suspect that, though Elodin involves optimization starting from the textual modality alone, each iteration guides the initial concept to the visual region of the cross-modal space due to the fact that one of the inputs to the similarity function (the generated image) is always a visual input.

6.2 Using namecons

Figure 4. Using the namecon



Source: the author (2023)

To use the result of the naming process, i.e. the namecon, I suggest a slightly different inference scheme than the usual one from inversion methods[12]. Instead of mapping the embedding to a developer-defined rare word (or, technically, “rare token"), allow the user to specify a *guiding concept* for each prompt.

With a guiding concept, each prompt follows a syntax as outlined below:

“<a bird | my_hawk> amidst white flowers”

In this example, !a bird" would be the guiding concept and “my_hawk" would refer to the namecon. The guiding concept in this initial example corresponds to the initial concept that was used to create the namecon; later examples in this section will develop on this basis.

To use a guiding concept, it is necessary to modify the inference script (recall that the namecon itself can be used without modification through any feature that loads embeddings). The modified script includes a parsing function that ignores the namecon at first, outputting, for example, “a bird amidst white flowers" as a text string. This is the text that is sent to the text encoder, which computes the sentence's embeddings as normal. That parsing function, however, also returns the position of the guiding concept in the prompt. In this example, it would return that, in the sentence “a bird amidst white flowers", the guiding concept "a bird" occupies the beginning of the sentence. The substring "a" would correspond to the first embedding and "bird" would correspond to the second embedding, as discussed in chapter 5, "Fundamentals". In technical terms, the parsing function would output the text string “a bird amidst white flowers" and the integers 1 and 2, which are the positions of "a" and "bird" in the phrase (since token 0 is always beginning-of-phrase). The positions are stored in a variable for later use.

After the text encoder outputs the usual embeddings, the inference script substitutes the embedding(s) corresponding to the guiding concept (“a bird") for the embedding(s) corresponding to the namecon. For that, it is necessary to know the position of the guiding concept in the original prompt. This is because the embeddings that come out of the text encoder are not easily interpretable, so it is more direct to substitute based on the position. Consider “<a bird | my_hawk> amidst white flowers" and another prompt such as “white flowers surrounding < a bird | my_hawk”. In the first case, the stored embedding from the namecon would replace rows 1 and 2 of the matrix that comes out of the text encoder. In the latter case, it would replace positions 4 and 5.

We employ this new inference method because the rare word employed by current inversion[12], even if apparently meaningless (such as a standard empty token), can

distort the inference process (since every word influences the embedding of all others). To illustrate this point, consider the results presented in Table 5, which use the namecon of a lava armor, but the guiding concept of a bird. All of them were generated in the same batch.

Table 5 <a bird | lava armor >on a frozen mountain peak

Source: the author (2023)

For some random seeds, the lava armor takes a bird-like shape. Even though the embeddings for the guiding concept (“a bird") have been replaced by the namecon, the context of the phrase (i.e. the other embeddings) are influenced by the presence of the guiding concept before the substitution takes place. In other words, even though the embeddings that are input to the image decoder correspond to "lava armor on a frozen mountain peak", a bird still appears in the images. This is because the embeddings for "on a frozen mountain peak" were originally created from the full phrase "a bird on a frozen mountain peak" before the embedding for "a bird" was replaced with the embedding for "lava armor". Therefore, the embeddings for "on a frozen mountain peak" still carry contextual information. To avoid problems in previous systems (for example, due to the use of the apparently meaningless word "sks", which resulted in images containing unintentional semi-automatic rifles), one may employ guiding concepts.

The aforementioned rare word method is equivalent to specifying a fixed guiding concept that one believes will not influence the results much. In contrast, our method enhances the prompter's control, allowing for an explicit choice of the guiding concept in each prompt. A mismatch between namecon and guiding concept can also be introduced intentionally, for example, by artists who are interested in producing images of birds wearing lava armor.

7 EXPERIMENTS

In the following text, "we" refers to the authors of the first Elodin pre-print[24]. The text in this chapter (outside this paragraph) has been reproduced exactly from that publication, except for chapter numbers and small corrections.

7.1 Experimental Apparatus

To assess how well Elodin solves the aforementioned concept coherence and concept cohesion issues, we perform qualitative side-by-side comparisons between images from an unaltered text-to-image pipeline (!control") and images from a pipeline modified with our method (!proposal"). We keep all configurations the same (the random seed, negative prompt, model weights, checkpoint, etc.), only changing the prompt. In the !proposal" setting, we replace each occurrence of the keywords associated with the target prompt by the corresponding namecon. For instance, we compare an image generated using the prompt !a yellow hawk amidst white flowers" with another generated using “<a bird | my_hawk> amidst white flowers". The prompts were chosen to demonstrate a wide range of visual concepts, such as textures, people, animals and objects.

For the qualitative evaluation (text setting), even though we do not enlist a large number of crowd workers, we make the generated images fully available in the supplementary material (no cherry-picking).
We provide a batch of 16 images with corresponding random number generator seeds per experiment.

For the quantitative evaluation (face id setting), we also perform analysis based on a face similarity score. We generate 100 images for both control and proposal. We then use the ArcFace similarity function[6] as a proxy to measure coherence between all pairs inside each group. The model used in this analysis is different from the one used to create the namecon in order to keep a fair comparison. If our hypothesis (that the use of namecons increases coherence in the generation of images of faces) is correct, then the average similarity among the proposal group should be higher than among the control group.

The control prompt we use for the quantitative experiments is “close-up shot of the face of a middle-aged blonde woman with green eyes and thin chin at the park. She is next to a big tree". The proposal prompt is “close-up shot of the face of <a woman | Lucy> at the park. She is next to a big tree”. For both of those, we employ the negative prompt “bad artist, low quality”.

For naming, we used SD 1.4 for the text setting (that is, texts as target concepts) and a finetuned version of SD 1.4 for the face id setting (that is, images as target concepts to generate more natural namecons). Learning rate was set to 4e-2 for text setting and 2e-2 for face id setting, batch size was 1. For inference, we used the Dreamshaper[9] model in the Stylized Lucy experiment, all others employed Stable Diffusion 1.4. Inference was performed through a popular tool, AUTOMATIC1111's WebUI[2]. The diffusion sampler used for naming was DDIM[37]. For inference, we used Ancestral Euler[11]. The similarity score in the text setting is based on CLIP[25], while the similarity score in the face id setting is based on facenet[36].

7.2 Results and Discussion

7.2.1 Qualitative

Table 6 ‘Bird’ experiment. Top row displays samples generated without the namecon.

Source: the author (2023)

Table 7 ‘Armor’ experiment. Top row displays samples generated without the namecon.

Source: the author (2023)

Some of the most clear examples from the supplementary are summarized in the following image comparisons. For each experiment, the top row shows the control configuration (no modification), while the bottom row shows the proposal configuration (ours), as discussed in this chapter. For each column, all parameters other than the prompt are the same, including the seed.

The Bird and Armor experiments highlight the increase of cohesion with Elodin. In Table 6, notice how the bird keeps its yellow color in the bottom row, white the flowers are much less yellow. In Table 7, notice how the mountain does not catch fire.

The Lucy and Cloth experiments highlight the increase in coherence. Notice how the person's facial appearance is kept the same in the bottom row (even through different hair colors). In Table 8, notice how the texture of the garment's fabric has a more uniform look from one image to the next.

Table 8 ‘Lucy’ experiment. Top row displays samples generated without the namecon.

Source: the author (2023)

Table 9 ‘Cloth’ experiment. Top row displays samples generated without the namecon.

Source: the author (2023)

It is possible to use the same namecon with different decoders as long as they share the same encoder. For example, one may generate images from the same namecon in different finetunings of Stable Diffusion version 1.x (such as SD 1.4 and SD 1.5), as demonstrated in 10. SD Version 2, however, uses OpenCLIP[17] instead of CLIP as an encoder, so we do not expect namecons to communicate between those versions.

Table 10 ‘Stylized Lucy’ experiment. Top row displays samples generated without the namecon.

Source: the author (2023)

Even though it is possible to generate faces in the text setting, the face id setting produces more natural pictures. For instance, in the text setting, the target concept “a blonde" may converge to a namecon that consistently results in images of blonde hair (that is, a hair close-up) instead of images of faces. We hypothesize this may stem from CLIP's low ability to distinguish faces (when compared to a specialized face recognition network such as Facenet). Recall that, as discussed previously, the generated face does not contain the identity of the face depicted in the image used as target concept.

7.2.1 Quantitative

As presented in Table 12, images created with namecons score higher on face similarity than images generated from a detailed regular prompt. For comparison, a value of cosine similarity above 0.45 (equivalent to an Euclidean distance below 1.1) would indicate pictures of the same person[36].

Table 11 Parameters for each experiment. We apply the following commomn negative prompt: “bad artist, low quality” in every experiment.

Experiment Name	Initial Concept	Target Concept	Guiding Concept	Control Prompt	Proposal Prompt	Similarity
Bird	“bird”	“a yellow hawk”	“a bird”	“a yellow hawk amidst white flowers”	“<a bird | my_hawk > amidst white flowers”	text (CLIP)
Armor	“fiery armor”	“armor made of re, lava, and very dark iron”	“armor”	“menacing warriors in armor made of fire, lava, and very dark iron atop a frozen mountain peak”	“menacing warriors in <armor | my_armor > atop a frozen mountain peak”	text (CLIP)
Cloth	“cloth”	“a very smooth	“cloth”	“a photo of a	“a photo of a	text (CLIP)

		celadon dress”		beautiful woman wearing a dress. The beautiful woman is wearing garment made of celadon cloth”	beautiful woman wearing a dress. The beautiful woman is wearing garment made of celadon <cloth | my_cloth >”	
Lucy	the name of a notorious actress	a face picture	“a woman”	“face of a blonde woman at the beach”	“face of <a woman | lucy > at the beach”	face (Facenet)
Stylized Lucy	same as the “Lucy” experiment	same as the “Lucy” experiment	same as the “Lucy” experiment	“a blonde woman at a business meeting”	“<a woman | lucy >at a business meeting”	face (Facenet)

Source: the author (2023)

Table 12 Similarity mean and standard deviation (std) within groups. Higher is better

	Mean	Std
Control	0.43	0.14
Proposal	0.71	0.15

Source: the author (2023)

8 CONCLUSION

Motivated by the challenge of fine-grained control in current text-to-image pipelines, the first contribution is to propose the generation of ``\textit{named concepts}'' (\textit{namecons}) in embedding space from minimal input data (such as a text prompt or a single picture of a face). \textit{Name Learning}, as I propose to call it, is another kind of Representation Learning (the first kind being Deep Learning). The second contribution is, then, theoretical and relates to the discussions in chapter 5, "Fundamentals".

The related experiments demonstrate the immediate consequences of the first contribution. As for the theoretical discussion, it cannot be directly tested, yet I hope that it will spark more development in the community.

Regarding future work, elodin can be generalized to other contexts, such as the previously mentioned color embeddings, or machine-generated artistic styles, or other modalities that do not involve text nor images.

Given that sections on ethics are common in papers nowadays, I ask two questions: Would you like to be treated as a machine by someone who holds the control of your rewards? How has everyone around you felt like in the past few years?

REFERENCES

[1] Pranav Aggarwal, Hareesh Ravi, Naveen Marri, Sachin Kelkar, F. Chen, Vinh Ngoc Khuc, Midhun Harikumar, Ritiz Tambi, Sudharshan Reddy Kakumanu, Purvak Lapsiya, Alvin Ghouas, Sarah Saber, Malavika Ramprasad, Baldo Faieta, and Ajinkya Kale. Controlled and conditional text to image generation with diffusion prior. ArXiv, abs/2302.11710, 2023.

[2] AUTOMATIC1111. Stable diffusion web ui, 2022.
https://github.com/ AUTOMATIC1111/stable-diffusion-webui [Accessed: March 2023].

[3] Andrew Brock, Jeff Donahue, and Karen Simonyan. Large scale gan training for high delity natural image synthesis. arXiv preprint arXiv:1809.11096, 2018.

[4] Tim Brooks, Aleksander Holynski, and Alexei A. Efros. Instructpix2pix: Learning to follow image editing instructions. ArXiv, abs/2211.09800, 2022.

[5] Niv Cohen, Rinon Gal, Eli A. Meirom, Gal Chechik, and Yuval Atzmon. "this is my unicorn, uffy": Personalizing frozen vision-language representations. In European Conference on Computer Vision, 2022.

[6] Jiankang Deng, Jia Guo, Niannan Xue, and Stefanos Zafeiriou. Arcface: Additive angular margin loss for deep face recognition. In Proceedings of the IEEE/CVF conference on computer vision and pattern recognition, pages 4690–4699, 2019.

[7] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. ArXiv, abs/1810.04805, 2019.

[8] Prafulla Dhariwal and Alexander Nichol. Diffusion models beat gans on image synthesis. Advances in neural information processing systems, 34:8780–8794, 2021.

[9] DreamShaper. Dreamshaper, 2023. https://civitai.com/models/4384/ dreamshaper [Accessed: March 2023].

[10] Patrick Esser, Robin Rombach, and Bjorn Ommer. Taming transformers for highresolution image synthesis. In Proceedings of the IEEE/CVF conference on computer vision and pattern recognition, pages 12873–12883, 2021.

[11] Hugging Face. Euler ancestral scheduler, 2022. https://huggingface.co/
docs/diffusers/api/schedulers/euler_ancestral [Accessed: March 2023].

[12] Rinon Gal, Yuval Alaluf, Yuval Atzmon, Or Patashnik, Amit H Bermano, Gal Chechik, and Daniel Cohen-Or. An image is worth one word: Personalizing text-to-image generation using textual inversion. arXiv preprint arXiv:2208.01618, 2022.

[13] Rinon Gal, Moab Arar, Yuval Atzmon, Amit H. Bermano, Gal Chechik, and Daniel Cohen-Or. Designing an encoder for fast personalization of text-to-image models. ArXiv, abs/2302.12228, 2023.

[14] Tao Ge, Jing Hu, Li Dong, Shaoguang Mao, Yanqiu Xia, Xun Wang, Siyi Chen, Furu Wei, and Si-Qing Chen. Extensible prompts for language models. ArXiv, abs/2212.00616, 2022.

[15] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. Advances in neural information processing systems, 27, 2014.

[16] Lianghua Huang, Di Chen, Yu Liu, Yujun Shen, Deli Zhao, and Jingren Zhou. Composer: Creative and controllable image synthesis with composable conditions. ArXiv, abs/2302.09778, 2023.

[17] Gabriel Ilharco, Mitchell Wortsman, Ross Wightman, Cade Gordon, Nicholas Carlini, Rohan Taori, Achal Dave, Vaishaal Shankar, Hongseok Namkoong, John Miller, Hannaneh Hajishirzi, Ali Farhadi, and Ludwig Schmidt. Openclip, July 2021.

[18] Hyeonho Jeong, Gihyun Kwon, and Jong-Chul Ye. Zero-shot generation of coherent storybook from plain text story using diffusion models. ArXiv, abs/2302.03900, 2023.

[19] Tero Karras, Samuli Laine, and Timo Aila. A style-based generator architecture for generative adversarial networks. In Proceedings of the IEEE/CVF conference on computer vision and pattern recognition, pages 4401–4410, 2019.

[20] Tero Karras, Samuli Laine, Miika Aittala, Janne Hellsten, Jaakko Lehtinen, and Timo Aila. Analyzing and improving the image quality of stylegan. In Proceedings of the IEEE/CVF conference on computer vision and pattern recognition, pages 8110–8119, 2020.

[21] Bahjat Kawar, Shiran Zada, Oran Lang, Omer Tov, Hui-Tang Chang, Tali Dekel, Inbar Mosseri, and Michal Irani. Imagic: Text-based real image editing with diffusion models. ArXiv, abs/2210.09276, 2022.

[22] Nupur Kumari, Bingliang Zhang, Richard Zhang, Eli Shechtman, and Jun-Yan Zhu. Multi-concept customization of text-to-image diffusion. arXiv, 2022.

[23] Weixin Liang, Yuhui Zhang, Yongchan Kwon, Serena Yeung, and James Y. Zou. Mind the gap: Understanding the modality gap in multi-modal contrastive representation learning. ArXiv, abs/2203.02053, 2022.

[24] Rodrigo Mello, Filipe Calegario, and Geber Ramalho. Elodin: Naming concepts in embedding spaces. arXiv preprint arXiv:2303.04001, 2023.

[25] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. Learning transferable visual models from natural language supervision. In International conference on machine learning, pages 8748–8763. PMLR, 2021.

[26] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. arXiv preprint arXiv:1511.06434, 2015.

[27] Colin Raffel, Minh-Thang Luong, Peter J. Liu, Ron J. Weiss, and Douglas Eck. Online and linear-time attention by enforcing monotonic alignments. In International Conference on Machine Learning, 2017.

[28] Aditya Ramesh, Prafulla Dhariwal, Alex Nichol, Casey Chu, and Mark Chen. Hierarchical text-conditional image generation with clip latents. ArXiv, abs/2204.06125, 2022.

[29] Aditya Ramesh, Prafulla Dhariwal, Alex Nichol, Casey Chu, and Mark Chen. Hierarchical text-conditional image generation with clip latents. arXiv preprint arXiv:2204.06125, 1(2):3, 2022.

[30] Robin Rombach, A. Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. Highresolution image synthesis with latent diffusion models. 2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), pages 10674–10685, 2021.

[31] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models, 2021.

[32] Nataniel Ruiz, Yuanzhen Li, Varun Jampani, Yael Pritch, Michael Rubinstein, and Kr Aberman. Dreambooth: Fine tuning text-to-image diffusion models for subject-driven generation. ArXiv, abs/2208.12242, 2022.

[33] Chitwan Saharia, William Chan, Saurabh Saxena, Lala Li, Jay Whang, Emily L. Denton, Seyed Kamyar Seyed Ghasemipour, Burcu Karagol Ayan, Seyedeh Sara Mahdavi, Raphael Gontijo Lopes,

Tim Salimans, Jonathan Ho, David J. Fleet, and Mohammad Norouzi. Photorealistic text-to-image diffusion models with deep language understanding. ArXiv, abs/2205.11487, 2022.

[34] Axel Sauer, Tero Karras, Samuli Laine, Andreas Geiger, and Timo Aila. Stylegant: Unlocking the power of gans for fast large-scale text-to-image synthesis. ArXiv, abs/2301.09515, 2023.

[35] Kevin Scharp. Replacing truth. Inquiry, 50(6):606–621, 2007.

[36] Florian Schroff, Dmitry Kalenichenko, and James Philbin. Facenet: A unied embedding for face recognition and clustering. In Proceedings of the IEEE conference on computer vision and pattern recognition, pages 815–823, 2015.

[37] Jiaming Song, Chenlin Meng, and Stefano Ermon. Denoising diffusion implicit models. ArXiv, abs/2010.02502, 2020.

[38] Andrey Voynov, Kr Aberman, and Daniel Cohen-Or. Sketch-guided text-to-image diffusion models. ArXiv, abs/2211.13752, 2022.

[39] Binxin Yang, Shuyang Gu, Bo Zhang, Ting Zhang, Xuejin Chen, Xiaoyan Sun, Dong Chen, and Fang Wen. Paint by example: Exemplar-based image editing with diffusion models. ArXiv, abs/2211.13227, 2022.

[40] Huahui Yi, Ziyuan Qin, Wei Xu, Miaotian Guo, Kun Wang, Shaoting Zhang, Kang Li, and Qicheng Lao. Cones: Concept embedding search for parameter efcient tuning large vision language models. arXiv preprint arXiv:2305.18993, 2023.

[41] Ge Yuan, Xiaodong Cun, Yong Zhang, Maomao Li, Chenyang Qi, Xintao Wang, Ying Shan, and Huicheng Zheng. Inserting anybody in diffusion models via celeb basis. arXiv preprint arXiv:2306.00926, 2023.

[42] Lvmin Zhang and Maneesh Agrawala. Adding conditional control to text-to-image diffusion models. ArXiv, abs/2302.05543, 2023.