EMBEDDING-BASED DETECTION OF AI-GENERATED IMAGES: A VECTOR SIMILARITY AND BLOCKCHAIN-BACKED APPROACH

Jitendra Sharma

A THESIS REPORT

Presented to the Faculty of Miami University in partial
fulfillment of the requirements
for the degree of

Master of Science

Department of Computer Science & Software Engineering

The Graduate School
Miami University
Oxford, Ohio

2025

Dr. Suman Bhunia, Advisor
Dr. Arthur Carvalho, Reader
Dr. Daniela Inclezan, Reader

ABSTRACT

The rapid advancement of generative AI and large language models (LLMs) has revolutionized various domains by enabling the creation of highly realistic and contextually relevant digital content. Verifying the integrity and origin of digital data to ensure it remains unaltered and genuine is crucial to maintaining trust and legality in digital media, a process known as digital content authentication. LLMs, such as ChatGPT and Stable Diffusion techniques, can produce images that are often indistinguishable from those created by humans, posing challenges to digital content authentication. This thesis addresses the critical need for an effective AI-generated image detection mechanism that can overcome both technical and ethical challenges. We present EmbedAIDetect, a blockchain-based AI image detection system that utilizes image embedding techniques and vector similarity search. We have compared and evaluated different embedding methods that are generalizable and resistant to significant image manipulations. The results of our experiments demonstrate an average accuracy of 95.67% in image classification using embedding techniques. The thesis presents a novel system that accepts image uploads from users, extracts their vector embeddings using a pre-trained model, compares them against a database of known embeddings, and verifies their integrity through hash checks on the blockchain network.

Table of Contents

List of Tables

List of Figures

Acknowledgements

I express my sincere gratitude to several individuals who have been instrumental in completing this thesis.

First, I am deeply grateful to my thesis advisor, Dr. Suman Bhunia, whose guidance, patience, and consistent support were vital throughout this research journey. Dr. Bhunia's expertise, meticulous attention to detail, and ability to set clear research directions and next steps provided me with the structure and focus needed to bring this work to fruition.

I extend my heartfelt appreciation to Dr. Arthur Carvalho, whose architectural vision laid the foundation for this research project. His technical insights and implementation feedback were invaluable in shaping the practical aspects of this work.

I am also profoundly grateful to Dr. Daniela Inclezan for her thoughtful and impactful feedback. Her suggestions and ethical support as a committee member meant a lot to me.

Together, my committee created a supportive and intellectually stimulating environment that allowed me to grow as a researcher. Their collective expertise and encouragement were the cornerstones of my academic development during this process.

Finally, I want to express my sincere gratitude to all the individuals who have supported me in various ways during the completion of this thesis. Their direct or indirect contributions have played an important role in my academic success. Thank you to everyone involved.

Chapter 1

Introduction

Generative AI and large language models (LLMs), such as ChatGPT [1], have made remarkable progress in recent years, transforming various industries by automating and enhancing creative processes. These models, powered by advanced neural network architectures, can generate human-like text and highly realistic images from simple prompts. With the release of DALL-E, Google's Imagen, Stable Diffusion, and Midjourney – diffusion models have revolutionized the field of image generation, inspiring creativity, and pushing the boundaries of machine learning [3]. These models generate a near-infinite variety of images from textual descriptions, ranging from photo-realistic to fantastical within seconds [4]. However, as these models become more sophisticated, their outputs become increasingly indistinguishable from human-created content, posing significant challenges in detecting AI-generated material.

1.1 Motivation

The proliferation of AI-generated content threatens artists and creators [5], as AI models can replicate the styles and techniques of human artists, raising concerns about intellectual property and originality [6]. Artists may struggle to protect their work from being copied or mimicked by AI, potentially undermining the value of original art [7]. In 2022, the Colorado State Fair art competition witnessed an unprecedented event in which an AI-generated image titled *Théâtre D'opéra Spatial* by Jason M. Allen won the first prize [8]. The picture was created using the diffusion model-based program *Midjourney* [2], which transforms text prompts into hyper-realistic artwork. This incident captivated the public and judges, sparked controversy in the art community, and raised questions about AI's role in creative fields. In addition to creative fields, AI-generated images contribute to online disinformation that threatens public discourse and trust. A notable example was the viral circulation of fake images depicting the arrest of US President Donald Trump [9]. These fabricated visuals demonstrated AI's sophisticated capability to create convincing but entirely fictional content.

The field of AI-generated image detection is rapidly evolving, yet no single solution has emerged that can universally detect images produced by all AI models. Current detectors are often specialized and capable of identifying images generated by specific models such as generative adversarial networks (GANs) or diffusion models, but struggle with newer versions or different types of models [10]. Tools like *AI or Not* [3] effectively detect images

[1] `https://openai.com/index/chatgpt/`
[2] `https://www.midjourney.com/`
[3] `https://www.aiornot.com/`

from Stable Diffusion 2, DALL-E 2, and GANs but face challenges with more advanced models like Stable Diffusion 3.5 [4] and DALL-E 3 [5]. Although convolutional neural network-based detectors perform well in some cases, they often fail against these advancements. A recent study by Jiang et al. has shown that watermarking techniques, while promising for their noninvasive approach to image authentication, remain insufficient against sophisticated evasion methods, underscoring the need for innovative detection approaches [11].

1.2 Open Challenges

Current detection methods struggle to determine whether an image, although resembling the original artwork, was generated by AI. The challenge lies in distinguishing between human and AI creativity while ensuring the preservation of artistic integrity and adherence to copyright standards. The following are the key challenges that our approach aims to address.

- **Generalization across generative models:** Detecting AI-generated images is challenging due to the variety of generative models. Existing detectors often work well with specific models like GANs or diffusion models but fail with newer models (e.g., Stable Diffusion 3 or DALL-E 3).
- **High computational cost and complexity:** Current detection models require significant computational resources for both training and deployment. This complexity makes them impractical for real-time use, particularly in smaller organizations or resource-constrained environments, underscoring the need for more efficient detection algorithms.
- **Robustness against image alterations:** AI-generated images are often manipulated (e.g., resizing, compression), which can reduce detection accuracy. Many current detectors fail when images are altered, making them less reliable in real-world use.

1.3 Proposed System

Based on the aforementioned challenges of distinguishing AI-generated images from real ones, we propose EmbedAIDetect, a hybrid system designed to classify and verify image origins. EmbedAIDetect allows users to upload an image, which is processed using a vision transformer model to generate its embedding. These embeddings are then used for two purposes: similarity-based classification through a vector database and tamper-proof verification via blockchain smart contracts. The system was developed through three prototype versions, starting with a blockchain-only model, followed by a vector database-based version, and culminating in a hybrid approach that combines the strengths of both. While the vector

[4] `https://stability.ai/news/introducing-stable-diffusion-3-5`
[5] `https://openai.com/index/dall-e-3/`

database enables semantic similarity comparisons, the blockchain ensures that classification results can be verified against an immutable ledger. The final prototype integrates these components into a unified web application. More details on the embedding pipeline, classification logic, and verification mechanisms are presented in the subsequent sections of this paper.

1.3.1 Contributions

The thesis research has made the following contributions:

1. Provides a resource-efficient, training-less AI image detection system that utilizes a similarity score to determine the authenticity of images.

2. Identifies and implements a suitable embedding technique that is generalizable and resistant to intentional manipulations and post-adversarial tampering attacks.

3. Integrate vector databases with embedding model and blockchain to maintain a tamper proof record of embeddings enhancing integrity of the detection system for real-time purposes.

1.3.2 Results

The EmbedAIDetect prototype was evaluated to assess its ability to classify images as either AI-generated or human-created, leveraging vector similarity-based classification and blockchain-based verification. A diverse set of image embeddings was used to evaluate the system performance, and key metrics such as precision, recall, and accuracy were measured. The evaluation also considered the efficiency of smart contract interactions, focusing on gas usage during blockchain operations.

The results demonstrated that the system achieved optimal performance, with an average accuracy across all models of 95.67% and resilience against adversarial image modifications, including geometric distortions and varying blur levels. In terms of blockchain efficiency, the system was found to be 2.7 times more gas-efficient when storing embedding hashes as integers (uint256) compared to using strings. However, the system's performance slightly degraded when handling real human facial images, where it occasionally misclassified them as AI-generated, underscoring the need for further refinement in distinguishing between human faces and synthetic counterparts.

Chapter 2

Background & Related Work

This chapter provides an overview of the background and related work essential for the thesis project. The background comprises four main topics: neural networks, generative AI, image generation, and image analysis techniques. Understanding these areas is crucial for developing a comprehensive detection system because they provide foundational knowledge on the underlying technologies and methodologies for generating and detecting AI-generated images. This knowledge allows for identifying the strengths and limitations of current approaches to address specific challenges effectively. This foundation ensures that our detection system can accurately and reliably differentiate between AI-generated images and original artworks in diverse and evolving scenarios.

2.1 Neural Networks

Neural networks are computational models inspired by the structure and function of the human brain. They consist of interconnected nodes, or neurons, organized in layers. Each neuron receives input signals, processes them through weighted connections, and produces an output that is passed to subsequent neurons. The perceptron is the fundamental building block of neural networks, which simulates a biological neuron's behavior by applying an activation function to the weighted sum of inputs to produce a binary output. This architecture allows neural networks to learn and model complex patterns in data through training, which involves adjusting the weights based on the error of the network predictions [12].

Neural networks learn through a process called backpropagation, in which the error is propagated backwards from the output layer to the input layer, and the weights are updated using gradient descent to minimize the error. This iterative process continues until the network's performance stabilizes, allowing it to make accurate predictions on new, unseen data. Neural networks have been applied to various tasks, including image recognition, speech recognition, and natural language processing, demonstrating their versatility and power to handle various types of data [13].

The subsequent sections present fundamental concepts underlying neural networks and deep neural networks. These concepts are the building blocks of many machine learning and artificial intelligence applications.

2.1.1 Deep Neural Networks

Deep neural networks (DNNs) extend the concept of neural networks by adding multiple hidden layers between the input and output layers. These additional layers enable DNNs to learn hierarchical representations of data, capturing more abstract and complex features at each successive layer. The depth of the network allows it to model intricate patterns and dependencies in data, making DNNs particularly effective for tasks such as image and speech recognition. Training DNNs requires large datasets and significant computational resources, but advances in hardware (e.g., GPUs) and optimization algorithms have made it feasible to train very deep networks [14].

One of the key breakthroughs in deep learning is the development of convolutional neural networks (CNNs) [15], a specialized type of DNN designed for processing grid-like data such as images. CNNs utilize convolutional layers that apply a series of filters to the input data, detecting local patterns and features such as edges, textures, and shapes. These convolutional layers are followed by pooling layers, which reduce the dimensionality of the data while preserving essential features, making the network more efficient and robust to variations in the input [16].

2.1.2 Recurrent Neural Networks

Recurrent Neural Networks (RNNs) are a type of neural network designed to handle sequential data by maintaining a hidden state that captures information about previous inputs. This feature makes RNNs particularly effective for tasks requiring contextual understanding over time, such as language processing, time series analysis, and sequence prediction [17]. Unlike traditional feedforward neural networks, RNNs have connections that form directed cycles, allowing them to retain the memory of previous states. This architecture enables RNNs to recognize patterns and dependencies in sequential data, making them particularly useful for complex temporal tasks.

In detecting AI-generated images, RNNs can be leveraged to analyze sequences of features extracted from images. Convolutional Neural Networks (CNNs) are typically used to extract spatial features from image patches. These features are then fed into an RNN to identify patterns that indicate the synthetic nature of the images. The ability of RNNs to remember previous states and recognize sequential dependencies makes them adept at distinguishing subtle artifacts that other models might miss [18]. Advanced RNN architectures like Long Short-Term Memory (LSTM) networks and Gated Recurrent Units (GRUs) are particularly effective, as they can maintain information over longer sequences, improving the detection accuracy of AI-generated images [19].

The subsequent sections introduces specialized and robust neural network architectures (Advanced Neural Networks) that have expanded the capabilities of deep learning models.

2.1.3 Convolutional Neural Networks (CNNs)

Convolutional neural networks (CNNs) are specifically designed to handle the spatial structure of images, making them highly effective for image recognition and generation tasks. A CNN typically consists of an input layer, multiple convolutional and pooling layers, fully connected layers, and an output layer. The convolutional layers apply filters to the input image, creating feature maps that highlight various aspects of the image. The pooling layers then downsample these feature maps, reducing their size while retaining important information. This hierarchical structure allows CNNs to capture increasingly complex patterns and features as the data progresses through the layers [16].

CNNs have revolutionized computer vision by achieving state-of-the-art performance on various image-related tasks. For example, the AlexNet architecture, which won the ImageNet Large Scale Visual Recognition Challenge in 2012, demonstrated the power of deep CNNs in accurately classifying images into thousands of categories. Subsequent architectures, such as VGGNet, GoogLeNet, and ResNet, have further advanced the field by introducing innovations in network depth, connectivity, and training techniques. These advancements have enabled CNNs to be applied in diverse applications, including object detection, facial recognition, medical image analysis, and autonomous driving [20, 21]

2.1.4 Transformers

Transformers are a type of deep learning model introduced by Vaswani et al. (2017) [22] in their seminal paper "Attention is All You Need." Originally designed for natural language processing tasks, transformers have since been adapted for various applications, including image generation and detection. The key component of transformers is the self-attention mechanism, which allows the model to weigh the importance of different input tokens (or parts of an image) when making predictions. This mechanism enables transformers to capture long-range dependencies and relationships within the data, making them highly effective for complex tasks. Unlike recurrent neural networks (RNNs), transformers handle sequential data without the need for recurrent layers, achieving parallel processing and significantly speeding up training. Since transformers do not inherently process data in sequence, they use positional encodings added to the embeddings at the input layer to account for the order of tokens in a sequence. These positional encodings ensure the model can take into account the position of each word or token in the sequence. The self-attention mechanism further allows the model to dynamically focus on different parts of the input sequence, weighing their significance when producing an output sequence.

In image generation, transformers have been used to develop models like Vision Transformers (ViTs) and generative models such as DALL·E. ViTs treat image patches as tokens and apply the transformer architecture to these tokens, allowing the model to learn spatial relationships and generate coherent images. The flexibility and scalability of transformers make them suitable for handling large-scale image datasets and generating high-quality images. Transformers' ability to integrate contextual information from different parts of the image also enhances their performance in image detection tasks, where understanding the

relationships between various image components is crucial [23].

Encoder

The encoder processes the input data (e.g., an image) and converts them into latent representations. The encoder takes in a sequence of tokens and produces a fixed-size vector representation of the entire sequence. It compresses input data information into the decoder's context or 'memory'. One of the most popular transformer encoder models is BERT (Bidirectional Encoder Representations from Transformers), introduced by Google in 2018. BERT is pre-trained on large amounts of text data and can be fine-tuned for a wide range of Natural Language Processing (NLP) tasks [24].

In image generation, the encoder processes an image to extract its features, which are then passed to the decoder to create a new image. An example is DALL·E, where the encoder processes textual descriptions, and the decoder generates corresponding images, effectively converting text into images [4].

Decoder

The decoder takes the latent representation generated by the encoder and uses it to produce the output data. This setup is adequate for tasks like image-to-image translation and image generation from text descriptions. The decoder takes in a fixed-size vector representation of the context. It uses it to generate a sequence of words one at a time, each being conditioned on the previously generated words.

One of the most popular transformer decoder models is GPT-3 (Generative Pre-trained Transformer 3), introduced by OpenAI in 2020. The GPT-3 is a massive language model that can generate human-like text in various styles and genres [25]. The ability to capture complex relationships between different parts of the input data enables the encoder-decoder architecture to generate coherent and contextually appropriate images. The encoder-decoder framework allows for flexibility in handling different input and output modalities, making it a versatile choice for various image-generation tasks [26].

Autoencoders

Autoencoders [27] are neural networks designed to learn efficient data representations by training the network to reconstruct input data. An autoencoder consists of an encoder that maps the input data to a latent space representation and a decoder that reconstructs the original data from this latent representation. The primary objective is to minimize the reconstruction error, ensuring that the latent representation captures the essential features of the input data. Autoencoders are used for various tasks, including image denoising, dimensionality reduction, and anomaly detection [28]. An autoencoder ensures that the latent space can capture most of the information from the dataset space by forcing it to output what was fed as input to the decoder.

Variational Autoencoders (VAEs)

Variational Autoencoders (VAEs) represent a significant advancement in generative models for their ability to capture essential low-dimensional data representations and generate new samples. First introduced by Diederik P. Kingma et al. in 2013, VAEs have emerged as a crucial method for both representation learning and generative modeling [29]. Unlike traditional autoencoders, which compress data to a point in the latent space, VAEs encode data into a distribution, typically a Gaussian. This probabilistic encoding allows the model to generate new data by sampling from the latent distribution, thus providing a smoother and more continuous latent space representation. The encoder in a VAE extracts the mean and variance of the latent variables from the data, while the decoder uses this information, along with Gaussian noise, to generate new samples. The core objective of a VAE is to optimize the Evidence Lower Bound (ELBO), which balances the reconstruction loss and the Kullback-Leibler (KL) divergence [30] between the learned latent distribution and a prior distribution. This optimization ensures that the model not only reconstructs the input data accurately but also generates plausible new instances that cover unseen samples in the input data, making VAEs particularly effective for tasks such as image generation and anomaly detection [29].

2.2 Generative AI

Generative AI is a subfield of artificial intelligence that focuses on creating new content rather than merely analyzing or processing existing data. This technology can produce various types of content, including text, imagery, audio, and synthetic data [31]. It leverages machine learning models, particularly neural networks, to generate outputs that are often indistinguishable from those created by humans. Generative AI has surged in popularity due to advancements in user interfaces that allow for the easy creation of high-quality content in seconds. These interfaces have democratized access to powerful generative capabilities, enabling users without deep technical expertise to harness AI's potential for content creation [32].

Neural networks are fundamental to generative AI and modeling and generating new data. Generative Adversarial Networks (GANs), introduced in 2014 by Ian Goodfellow [18], have also played a crucial role in advancing the capabilities of generative AI, especially in natural language processing (NLP). They enable the training of large language models that generate coherent and contextually relevant text. Variational Autoencoders (VAEs) [29] encode input data into a latent space and then decode it to create new data, making them particularly useful for producing high-quality images and other complex data types. These advancements have collectively propelled generative AI to create highly realistic and varied forms of content, from text and images to audio and video.

Generative AI has experienced remarkable advancements since 2020, driven by developments in Large Language Models (LLMs) and multimodal AI applications. In 2020, OpenAI released GPT-3, a groundbreaking generative pre-trained transformer model with 175 billion

parameters. This model enhanced language generation capabilities, producing more coherent and contextually appropriate text (OpenAI, 2020). The following year, OpenAI introduced DALL-E, a model capable of creating images from textual descriptions [33]. The year 2022 marked a pivotal moment for generative AI as it began to gain mainstream adoption. In November, OpenAI launched ChatGPT, based on the GPT-3.5 architecture. ChatGPT quickly gained popularity for its ability to generate human-like conversational text, making it applicable across various applications such as customer service and content creation [34]. Google also contributed to these advancements by introducing MUM (Multitask Unified Model), a transformer-based model designed to handle complex queries and understand text across 75 languages [35]. In 2023, OpenAI released GPT-4, which improved GPT-3 by enabling more sophisticated content generation and data analysis tasks [36]. Google introduced Gemini (formerly Bard), a public-facing chatbot built on its LaMDA family of large language models. Despite initial inaccuracies, Google refined Gemini, integrating it with its most advanced LLM, PaLM 2, to enhance its efficiency and accuracy [37]. Also, Microsoft integrated a version of GPT into its Bing search engine, improving search capabilities with AI-driven content generation [38].

2.2.1 AI-Generated Images

AI-generated images are visual content created by artificial intelligence systems, typically through machine learning algorithms. These images are produced using models trained on vast datasets of existing images, allowing the AI to learn patterns, textures, and structures. The most popular models for generating images include Generative Adversarial Networks (GANs), Variational Autoencoders (VAEs), Transformer-based models like DALL·E, and Diffusion Models (DMs). These technologies enable the creation of images ranging from highly realistic photographs to abstract and artistic compositions as shown in Figure 2.1. The algorithms are developed to mimic the natural world's visual intricacies, creating images indistinguishable from those captured by human photographers [39, 40].

Importance in the Contemporary Digital Landscape

The emergence of AI-generated images represents a significant advancement in digital technology, impacting numerous sectors such as entertainment, advertising, and content creation. It is estimated that over 250 million companies (77%) are using or exploring AI in some capacity within their operations, and this trend is expected to continue. In entertainment, these images are used to generate realistic visual effects, create virtual characters, and design game environments. In advertising, AI-generated images allow for the rapid production of diverse and tailored visuals for various marketing campaigns, enhancing engagement and personalization. The ability to generate high-quality images quickly and cost-effectively has also facilitated new forms of artistic expression, where artists can collaborate with AI to explore innovative styles and concepts. As a result, AI-generated images are revolutionizing fields such as digital art, advertising, and media production, making high-quality visual content more accessible and versatile [41, 42].

Figure 2.1: An example of real images (top) and corresponding AI-generated images (bottom) from popular AI image generators showcasing advancements in digital image synthesis

However, the rise of AI-generated images also brings several ethical and practical challenges, including concerns about authenticity and potential misuse in creating misleading or harmful content [43, 44]. The ease with which AI image generator tools manipulate images to create misleading content threatens media credibility and public trust. AI-generated photos can propagate false narratives, misrepresent actual events, and alter public perception, impacting democratic processes and societal harmony. Also, the unauthorized use of personal images from social media can lead to privacy breaches and identity theft, as AI can forge credible (deepfakes) fake identities. Ethically, the non-consensual use of an individual's likeness raises concerns about ownership and control. These issues highlight the need for robust detection and authentication methods, stringent regulations, and ethical guidelines to ensure responsible and fair use of AI technologies in image generation.

2.3 Image Generation Models

This section explores various image generation models. Each model uses different techniques to synthesize new visual content from training data. As we delve deeper into this section, the focus shifts to Diffusion Models. This cutting-edge subclass utilizes stochastic processes to construct images incrementally, and this approach has set new benchmarks for image quality.

2.3.1 Generative Adversarial Networks (GANs)

Generative Adversarial Networks, commonly called GANs, are a revolutionary class of AI models for image generation introduced by Ian Goodfellow and his colleagues in 2014 [18]. GANs consist of two competing neural networks, a generator and a discriminator, which are trained simultaneously through an adversarial process. The generator creates fake samples from random input vectors, crafting new data points from scratch. In contrast, the discriminator acts as a binary classifier, evaluating whether the samples it receives are real (from the dataset) or fake (from the generator). The adversarial nature of this setup comes from the continuous competition between these two networks, similar to a zero-sum game where the generator aims to produce indistinguishable fake samples, and the discriminator strives to identify them correctly. An example of GAN architecture is shown in the Figure 2.2.

The training process of GANs is iterative and involves both networks learning from each other. When the discriminator accurately classifies a sample, it "wins," prompting the generator to adjust its parameters to create more realistic samples. Conversely, if the generator successfully fools the discriminator, the generator "wins," leading to the discriminator refining its ability to distinguish real from fake. This dynamic process ensures that both the generator and the discriminator improve over time, with the ultimate goal being that the generator produces highly realistic images that are challenging for the discriminator—and even humans—to distinguish from real images.

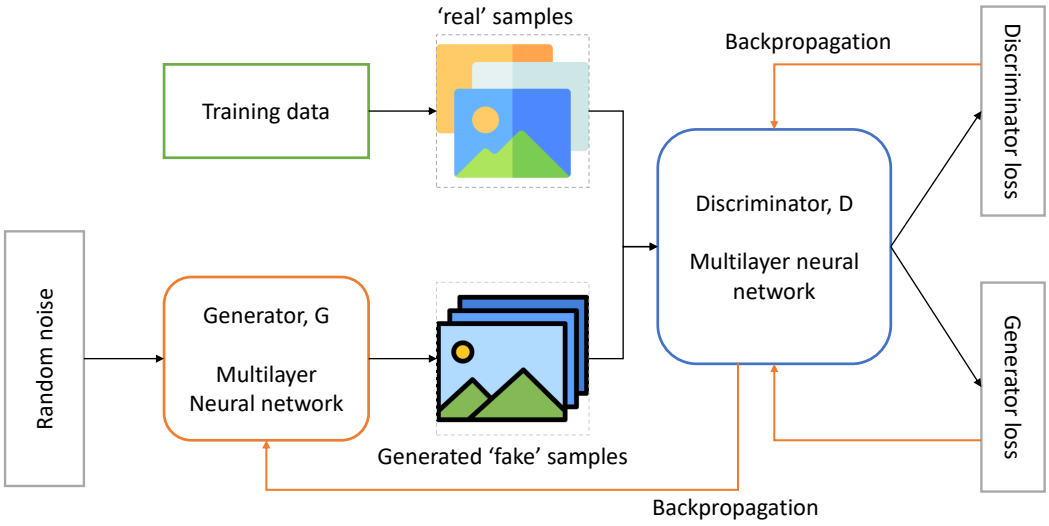


Figure 2.2: Example of a GAN Architecture

For the discriminator to function effectively, it needs a reference for what real images look like, provided through a labeled dataset. During training, the discriminator is presented with both real images (labeled as real) and fake images generated by the generator (labeled as fake). This ground truth enables the discriminator to learn to differentiate between real and generated images more accurately. Simultaneously, the generator receives feedback on how

convincingly it has produced fake images, using this information to refine its techniques. This feedback loop drives the continuous enhancement of both networks, making GANs a powerful tool for generating realistic images.

Several advancements have been made in GAN architectures to enhance their performance. The Deep Convolutional GAN (DCGAN) introduced by Radford et al. [19] utilizes deep convolutional layers to improve the quality and stability of image generation. DCGANs have been particularly successful in generating high-resolution images. Another notable variant, StyleGAN, developed by Karras et al. [45], incorporates a style-based architecture that allows for fine-grained control over the visual features of the generated images. StyleGAN has been widely recognized for its ability to produce highly detailed and realistic images [19, 45]. Additionally, BigGAN by Brock et al [46]. leverages large batch sizes and extensive datasets to generate images with greater diversity and detail.

In image synthesis, GANs are used to create photorealistic images from scratch. They are also employed in image editing tasks, such as inpainting, where missing parts of an image are generated to complete it. Furthermore, GANs are utilized in super-resolution to enhance the resolution of pictures and in domain adaptation to transfer styles between different image domains. These applications demonstrate the versatility and potential of GANs in advancing the field of image generation and manipulation [47, 48].

2.3.2 Autoregressive Models

Autoregressive models [49] generate images by predicting one pixel at a time, conditioned on the previously generated pixels. This sequential process ensures that each pixel is generated in the context of the preceding ones, allowing for the creation of coherent and high-quality images. The key idea behind autoregressive models is to model the probability distribution of each pixel given the values of all previous pixels, which makes them particularly effective for tasks requiring fine-grained detail and consistency. The probability of the entire image is computed as the product of the conditional probabilities of each individual pixel. The probability of each pixel's intensity is conditioned on all previously generated pixels, as expressed in the following equation:

$$p(\mathbf{x}) = \prod_{i=1}^{n^2} p(x_i | x_1, \ldots, x_{i-1})$$

This implies that to generate a specific pixel x_i in the image, it is necessary to consider the intensity values of all previously generated pixels.

PixelCNN, introduced by van den Oord et al. [49], is a notable autoregressive model that uses convolutional neural networks to model the conditional distributions of pixels. PixelCNN employs masked convolutions to ensure that the prediction of each pixel only depends on the pixels that have already been generated, preserving the autoregressive nature of the model. Another significant model, PixelSNAIL [50], enhances PixelCNN by incorporating self-attention mechanisms, which allow the model to capture long-range dependencies and improve the quality of generated images [50].

These models are used in high-fidelity image generation, where the goal is to produce images with fine details and high visual quality. They are also employed in image completion tasks, where missing parts of an image are predicted and filled in to create a complete image. Also, autoregressive models are used in image enhancement tasks, such as super-resolution, where low-resolution images are upscaled to higher resolutions while preserving details. These applications highlight the strengths of autoregressive models in generating high-quality images with detailed structure and texture [51].

2.3.3 Neural Style Transfer (NST)

Neural Style Transfer (NST) is a deep learning technique that blends the content of one image with the artistic style of another, creating a novel piece of artwork. Introduced by Gatys et al. [52] the method utilizes a pre-trained convolutional neural network (CNN) to analyze and manipulate the visual attributes of images. By borrowing the stylistic elements from one image and applying them to another, NST synthesizes new images that amalgamate the content and style of the input images, resulting in visually captivating outputs.

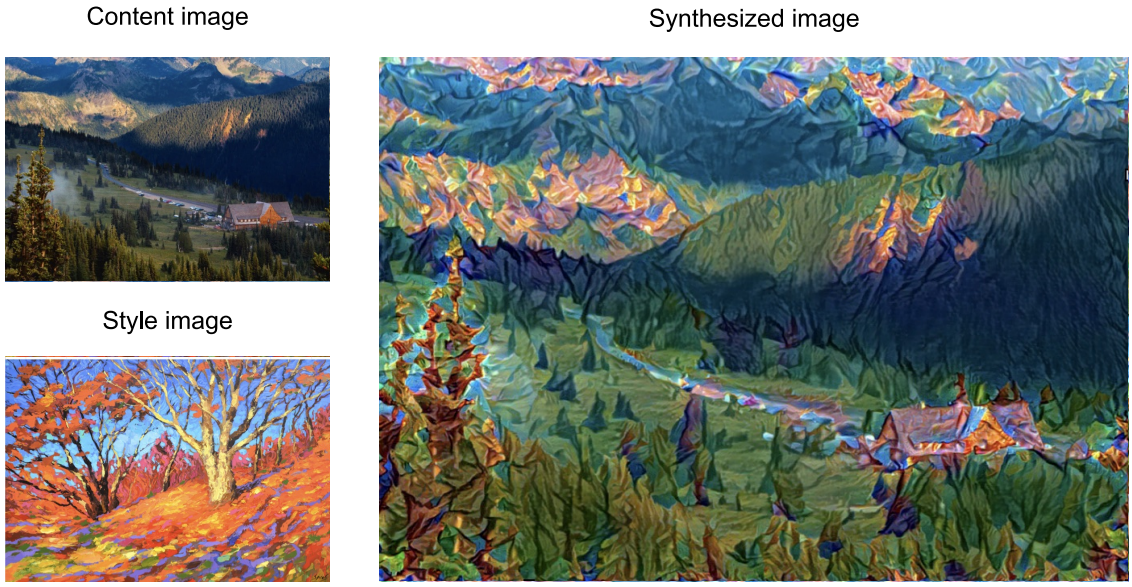


Figure 2.3: Given content and style images, the style transfer generates a synthesized image

The process of NST involves three primary images: the content image, the style image, and the generated image. The content image is the one whose structural elements and objects need to be retained. The style image provides the artistic features, such as textures, patterns, and colors, that need to be imposed on the content image. Initially, the generated image can be a random noise image or a copy of the content image. This generated image is iteratively modified to blend the content of the content image with the style of the style image, as illustrated in Figure 2.3. Here, the content image is a landscape photograph from Mount Rainier National Park, and the style image is an oil painting of autumn oak trees [53]. The

resulting synthesized image retains the main shapes of the content image while adopting the vivid brush strokes and colors of the style image. This process is typically achieved using a pre-trained convolutional neural network (CNN), such as VGG-19, which extracts hierarchical features from content and style images.

NST defines three primary loss functions to guide the blending process: content loss, style loss, and total variation loss. Content loss ensures that the generated image retains recognizable features of the content image by measuring the mean squared error between the feature maps of the generated and content images. Style loss quantifies the differences in textures and patterns and is calculated as the mean squared error between the Gram matrices of the feature maps of the generated image and the style image. Total variation loss helps to reduce noise in the generated image, promoting smoother transitions. By minimizing these losses through iterative optimization, NST effectively merges the content and style into a synthesized image that reflects the content image's structure and the style image's artistic style.

Several improvements and variants of neural style transfer have been developed to enhance its efficiency and flexibility. Fast Neural Style Transfer methods, such as those introduced by Johnson et al. [54], use feed-forward networks to achieve real-time style transfer. Adaptive Instance Normalization (AdaIN) [55] further improves flexibility by allowing the model to apply multiple styles in a single network, adjusting the instance normalization parameters according to the style. These advancements have expanded the applications of neural style transfer, enabling its use in real-time video processing, artistic image creation, and various content creation tasks.

2.3.4 Diffusion Models (DMs)

Diffusion Models [56] are generative models that learn from data during training and generate similar examples based on what they have learned. Inspired by non-equilibrium thermodynamics, these models employ a Markov chain of diffusion steps to slowly add random noise to data and then learn to reverse this process to construct desired data samples from the noise. Unlike VAEs or flow models, diffusion models are learned with a fixed procedure and have high-dimensional latent variables. Several diffusion-based generative models have been proposed with similar ideas underneath, including diffusion probabilistic models [56], noise-conditioned score network (NCSN) [57], and denoising diffusion probabilistic models (DDPM) [1].

Forward Diffusion Process

The forward diffusion process involves incrementally adding Gaussian noise to an input image over a series of T steps. Initially, at step 0, we start with the original image. At each subsequent step, the image is progressively corrupted by the noise, resulting in an image that becomes increasingly noisy. By the final step T, the image has been transformed into pure noise, effectively losing all original information. This transformation maps the clear image into noise space, setting the stage for the reverse process.

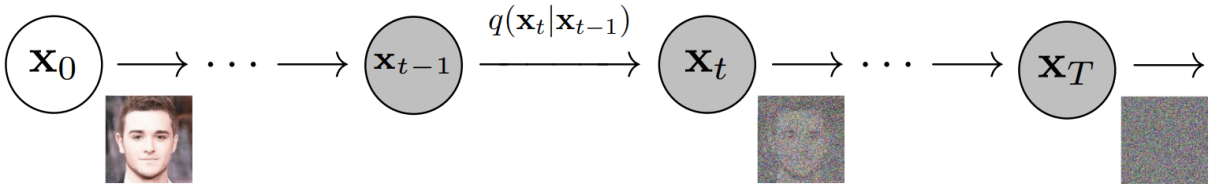


Figure 2.4: Forward diffusion process [1]

To formalize this process, it is modeled as a fixed Markov chain with T steps. In this chain, the image at timestep t transitions to the next state at timestep $t+1$. Each step in this sequence depends only on the preceding step, allowing for a closed-form solution to obtain the corrupted image at any specific timestep t without requiring iterative computation. The mathematical formulation is as follows:

$$q(x_{1:T}|x_0) := \prod_{t=1}^{T} q(x_t|x_{t-1}), \quad q(x_t|x_{t-1}) := \mathcal{N}(x_t; \sqrt{1-\beta_t}x_{t-1}, \beta_t\mathbf{I}) \tag{2.1}$$

$$q(x_t|x_0) = \mathcal{N}(x_t; \sqrt{\bar{\alpha}_t}x_0, (1-\bar{\alpha}_t)\mathbf{I}) \tag{2.2}$$

This closed-form expression enables direct sampling of x_t at any timestep, accelerating the forward diffusion process.

Schedulers

In diffusion models, the noise addition at each step follows a deliberate pattern determined by a scheduler. The scheduler specifies the amount of noise to be added at each step. In the original Denoising Diffusion Probabilistic Models (DDPM) paper [1], a linear noise schedule ranging β_t from 0.0001 at timestep 0 to 0.02 at timestep T is defined. However, several alternatives have gained popularity, such as the cosine schedule introduced in the Improved Denoising Diffusion Probabilistic Models paper by Nichol, A., & Dhariwal, P. (2021) [2] provides a smoother degradation, maintaining more image information at later steps, which results in higher quality and more stable generation.

Figure 2.5: Comparison of Linear and Cosine schedulers [2]

For example, the Figure 2.5 shows the difference between using linear and cosine schedules for the forward diffusion process. The first row displays a linear schedule, while the second

row demonstrates the improved cosine schedule.

The cosine schedule performs better than the linear schedule. A linear schedule may result in a rapid loss of information in the input image, leading to an abrupt diffusion process. In contrast, the cosine schedule provides a smoother degradation, allowing the later steps to operate on images that are not entirely overwhelmed by noise. This smoother progression results in higher quality and more stable image generation.

Reverse Diffusion Process

In contrast to the forward diffusion process, the reverse diffusion process presents a computational challenge, as the formulation of $q(x_{t-1}|x_t)$ becomes incomputable. To tackle these challenges, deep learning models are utilized to approximate the reverse diffusion process using a neural network.

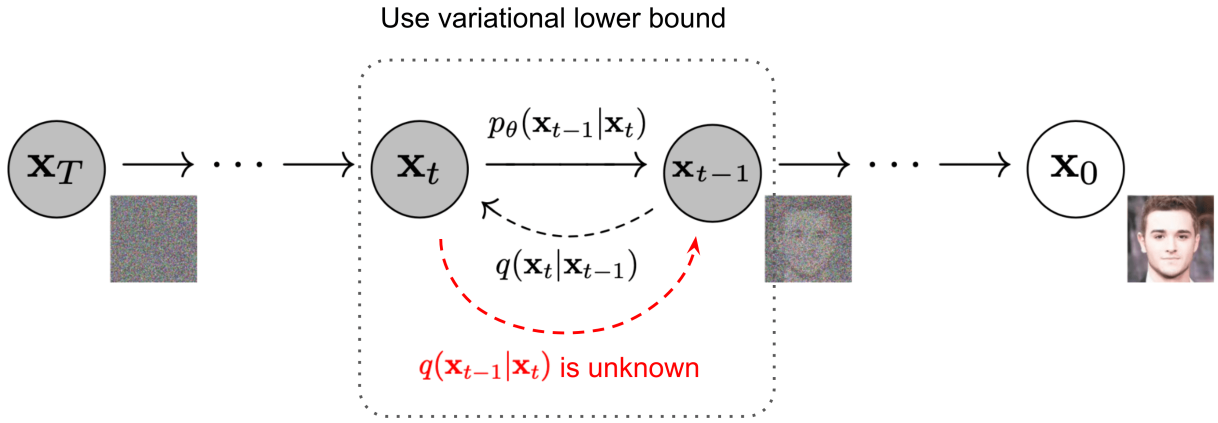


Figure 2.6: Reverse diffusion process [1]

The main task of this neural network is to predict the total noise present in the image at a given timestep t. By comparing this prediction with the actual noise added to the image, the network can be trained. During inference, these networks predict the total noise present at timestep t and then remove a fraction of this noise according to the scheduler used, thereby refining the image incrementally.

Empirical evidence suggests that instead of attempting to retrieve the original image in a single step by predicting the noise at timestep t, it is more stable to take smaller steps. This approach involves removing only a fraction of the noise present at timestep t to obtain the image at timestep $t-1$. This incremental approach ensures greater stability and higher-quality image reconstruction.

Unconditioned and Conditioned Generation

Unconditioned image generation refers to the process where a model generates images without any specific input conditions or constraints. In this approach, the model learns

the underlying distribution of the training data and creates new images that follow this learned distribution. Essentially, the model converts noise into any random representative data sample, producing an image of any nature. The generation process is neither controlled nor guided, allowing the model to create diverse and varied outputs.

Conditioned image generation involves generating images based on specific input conditions or constraints. These conditions can take various forms, such as class labels, textual descriptions, or other images. The goal is to generate images that adhere to the given conditions while maintaining high quality and diversity. In conditioned image generation, the model receives additional information during the training and inference stages, guiding the generation process to ensure that the output images match the specified conditions. This type of image generation is guided or controlled, allowing the model to produce specific sets of images based on the provided conditions. For instance, in text-to-image generation (text2img) or class-conditional generation (like in Conditional GANs), the additional information directs the model to generate images that align with the input text or class labels.

Stable Diffusion

Stable Diffusion is a deep learning, text-to-image model released in 2022 based on diffusion techniques. This model represents an advancement in generative artificial intelligence, developed by Stability AI, and is considered a part of the current AI boom. Primarily used to generate detailed images from text descriptions, Stable Diffusion can also perform tasks such as inpainting, outpainting, and image-to-image translations guided by text prompts [58]. Its development involved collaboration between the CompVis Group at Ludwig Maximilian University of Munich and the company Runway [59], with computational resources donated by Stability AI and training data from various non-profit organizations.

Stable Diffusion operates as a latent diffusion model (LDM), a type of deep generative artificial neural network. Unlike previous proprietary models like DALL-E and Midjourney, Stable Diffusion's code and model weights have been made publicly available [60], and it can run on most consumer hardware equipped with a modest GPU with at least 4 GB VRAM. This openness marks a departure from earlier models that were accessible only via cloud services, making powerful AI tools more accessible to a broader audience.

Stable Diffusion employs a three-part architecture: a variational autoencoder (VAE), a U-Net, and an optional text encoder. The VAE encoder compresses the image into a latent space, capturing its semantic meaning. Gaussian noise is iteratively added to this compressed representation during the forward diffusion process. The U-Net, built on a ResNet backbone, denoises the latent representation during the reverse diffusion process to obtain a clear image. The final image is generated by the VAE decoder, which converts the latent representation back into pixel space. Text conditioning is facilitated by the CLIP ViT-L/14 text encoder, which transforms text prompts into an embedding space [61].

Stable Diffusion was trained on LAION-5B, a publicly available dataset consisting of 5 billion image-text pairs. The dataset was curated by filtering based on language, resolution, watermark presence, and aesthetic scores. The model's training involved 256 Nvidia A100 GPUs on Amazon Web Services, totaling 150,000 GPU-hours at a cost of $600,000. The

latest version, SD3 [62], was trained at a significantly higher cost of around $10 million, reflecting the increasing complexity and capabilities of the model [63].

2.4 Image Analysis Techniques

This section delves into the techniques essential for analyzing and detecting AI-generated images. These techniques form the backbone of the proposed detection system, enabling efficient identification of AI-generated content. First, embedding techniques are explored, which transform images into mathematical representations. Next, various image similarity measures that quantify the likeness between images are examined. Finally, vector databases are discussed, which facilitate the storage and rapid retrieval of image embeddings. By understanding and integrating these image analysis techniques, a robust system capable of accurately distinguishing AI-generated images from original artworks can be developed.

2.4.1 Embedding

The process of representing the real world as data in a computer is called embedding and is necessary before the real world can be analyzed and used in applications. Embeddings represent data—almost any kind of data, like text, images, videos, users, music, etc.—as points in space where the locations of those points are semantically meaningful. For example, Word2Vec [64], a technique invented by Google in 2013, embeds words into an n-dimensional coordinate system where semantically similar words cluster together. This concept extends to other data types: for example, similar-sounding songs will be plotted near each other in a song-embedding space, and similar-looking images will be plotted near each other in an image-embedding space.

Embeddings are used due to their ability to facilitate finding similar data points through nearest neighbor search and compute numerical similarity scores between data points. This calculation often uses Euclidean distance, dot product, and cosine similarity measures. Embeddings support applications like duplicate detection and facial recognition, where pictures are embedded, and similarity scores determine if two images represent the same person. High similarity scores in an embedding space can also indicate near-duplicate photos. Additionally, embeddings can assist in typo correction by clustering common misspellings near the correct word in the embedding space.

Vector Embedding

Vector embeddings are powerful mathematical representations used to encode complex data, such as images, words, or sentences, into lower-dimensional vectors while preserving their intrinsic properties and relationships. These numerical representations transform various data types, including non-mathematical data, into arrays of numbers that machine learning models can process. By placing semantically similar items close together in space and dissimilar items far apart, vector embeddings enable models to detect patterns, group similar items,

find logical relationships, and make predictions. They can be produced for different kinds of information—words, phrases, sentences, images, nodes in a network, etc. Generating these embeddings involves an embedding model that transforms inputs into numerical vectors, as illustrated in Figure 2.7. This technique is fundamental in various machine learning and computer vision tasks because it allows for more efficient comparison and analysis of high-dimensional data. Created using techniques like Convolutional Neural Networks (CNNs) and other deep learning architectures, embeddings map data to a continuous vector space, maintaining semantic notions of similarity and dissimilarity. The quality of embedding is important as it directly impacts the performance of downstream tasks [64], so choosing an embedding model and training data are critical components of any AI system that relies on this technique.

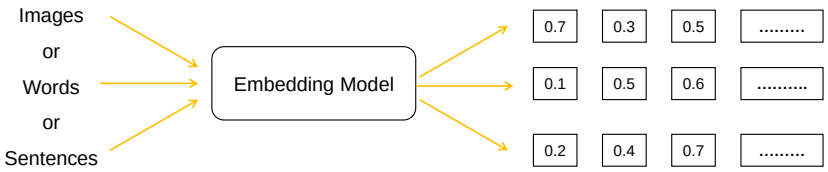


Figure 2.7: An embedding model transforms input data—such as images, words, or sentences into fixed-length numerical vectors

Image Embedding

Image embeddings are vector representations of images that capture their essential features and semantics in a lower-dimensional space. These embeddings are widely used in computer vision tasks such as image classification, object detection, image retrieval, and image similarity. By transforming images into numerical representations, embeddings enable efficient and effective analysis, comparison, and manipulation of visual data.

To understand how images are transformed into embeddings, consider an image with a width and height of 3 pixels each. As shown in Figure 2.8, this image can be decomposed into three primary colors: Red, Green, and Blue. Each color can be represented using a 2-D matrix, where each cell in the matrix contains a pixel value that indicates the color saturation (or brightness). For instance, the Red matrix has cells with values ranging from 0 to 255, each representing a different shade of red. Therefore, for a 3 x 3 image, we have:

A Red matrix with dimensions 3 x 3 = 9 pixels A Green matrix with dimensions 3 x 3 = 9 pixels A Blue matrix with dimensions 3 x 3 = 9 pixels Hence, the entire image is represented by a matrix with dimensions width x height x components = 3 x 3 x 3 = 27 pixels. In practice, image sizes are usually much larger than 3 x 3. For instance, a 1080p image has dimensions of 1920 x 1080 pixels, requiring 1920 x 1080 x 3 = 6,220,800 pixels.

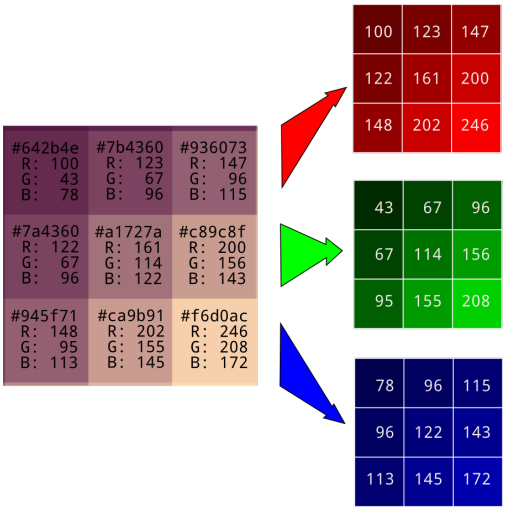


Figure 2.8: Representation of a 3 by 3 image using RGB matrices

Image embeddings are derived from image models trained on large datasets to learn how to extract meaningful information. These models include convolutional neural networks (CNNs), autoencoders, and generative adversarial networks (GANs), each having its own advantages depending on the task and data. An image embedding is a vector of numbers representing an image in a high-dimensional space. For example, an image of a cat can be embedded as a vector of 384 numbers, such as [0.12, -0.34, ..., 0.05]. Each number in the vector corresponds to a feature or attribute of the image, such as color, shape, or texture. This vector captures the essence of the image and allows comparison with other images using mathematical operations.

2.4.2 Embedding Models

Embedding models are algorithms that convert high-dimensional data into low-dimensional vectors, or embeddings, to make it easier for machine learning (ML) models to process. For image detection, models like ResNet, CLIP, and EfficientNet transform images into dense vectors that capture essential features and patterns.

Residual Network (ResNet) Embeddings

ResNet, introduced by He et al. [21] in "Deep Residual Learning for Image Recognition" (2015), captures high-level features of an image by passing it through a series of layers. ResNet utilizes residual blocks, which include convolutional layers, batch normalization, ReLU activation, and shortcut connections that allow input to be added directly to the output. This design helps optimize the learning of residual functions, facilitating the training of very deep networks without suffering from the vanishing gradient problem.

The process to obtain embeddings begins with resizing and normalizing the input image to match the expected size (typically 224×224 pixels). The image is then processed through an

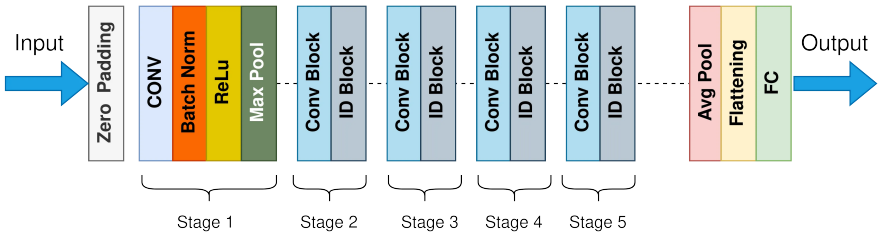


Figure 2.9: ResNet 50 model architecture

initial convolutional layer, followed by batch normalization and ReLU activation, capturing basic features. The image continues through multiple residual blocks, which apply filters to extract increasingly complex features, normalize the output, apply non-linearity, and use shortcut connections for learning residual functions. After passing through the residual blocks, the feature map's spatial dimensions are reduced, but its depth is increased. A global average pooling layer is applied to reduce each feature map to a single value, resulting in a feature vector. This output, the embedding, represents high-level, abstracted features of the image, typically with dimensions equal to the number of filters in the last convolutional layer.

In ResNet-50 as shown in Figure 2.9, a popular variant with 50 layers, the process involves bottleneck residual blocks that use 1×1 convolutions to reduce and restore dimensions, enhancing efficiency while maintaining performance. The final embedding from ResNet-50, obtained from the global average pooling layer, is a high-dimensional vector (typically 2048 dimensions) that encapsulates essential image features. This embedding is useful for tasks like image classification, retrieval, and detecting AI-generated images.

Contrastive Language-Image Pretraining

CLIP [65] is a multimodal embedding model developed by OpenAI and released in 2019. It was trained using over 400 million pairs of images and text. It can be instructed in natural language to predict the most relevant text snippet given an image without directly optimizing for the task, similar to the zero-shot capabilities of GPT-2 and 3. Both image and text embeddings can be calculated with CLIP, which can be used to compare image embeddings and text embeddings. CLIP is a zero-shot model, which means the model does not require any fine-tuning to calculate an embedding. For classification tasks, it is necessary to calculate text embedding for each potential category and image embedding to classify the image. Then, each text embedding is compared to the image embedding using a distance metric like cosine similarity. The text embedding with the highest similarity is the label most related to the image.

CLIP embeddings are used to understand and link images and text, capturing the semantic essence of both visual and textual inputs. The embeddings are generated using dual encoders: one for images and another for text. The contrastive pre-training approach used by CLIP is shown in Figure 2.10. The text encoder processes a variety of text descriptions, cre-

ating text embeddings $(T_1, T_2, \ldots, T_N)$. The image encoder processes corresponding images, creating embeddings $(I_1, I_2, \ldots, I_N)$. The embeddings are compared in a shared embedding space, where the model learns to associate matching pairs (e.g., I_1 with T_1) and to distinguish them from non-matching pairs (e.g., I_1 with $T_2, T_3, \ldots, T_N$). This training paradigm allows the model to learn generalized representations of both images and text, facilitating zero-shot transfer capabilities across various tasks.

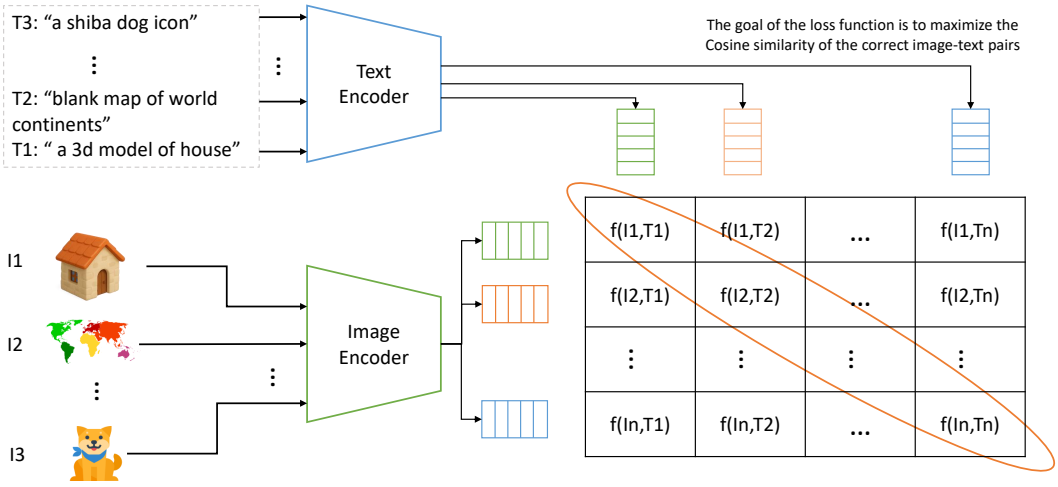


Figure 2.10: Overview of CLIP model

2.4.3 Image Similarity

Image similarity refers to the degree to which two images are alike, quantified using various metrics that compare pixel values, structural features, and high-level semantic content. Traditional methods like Mean Squared Error (MSE) and Peak Signal-to-Noise Ratio (PSNR) focus on pixel-level differences, whereas advanced techniques like the Structural Similarity Index (SSIM) consider perceptual and structural aspects of images [66]. Advanced similarity measures leverage deep learning-based embeddings to capture high-level features invariant to transformations like scaling, rotation, and noise, which are crucial for detecting AI-generated images. Embeddings from models like ResNet or CLIP can be compared using cosine similarity to determine how closely related two images are in terms of content and style, enhancing robustness and accuracy in image similarity assessments.

Detecting AI-generated images typically involves comparing the suspected image against a database of known real and synthetic images using various metrics and algorithms. Techniques such as SSIM [66] and pixel-wise feature extraction identify subtle differences between real and synthetic images. Research shows that cosine similarity can effectively distinguish

between real and AI-generated content, highlighting the potential of similarity-based methods in enhancing detection accuracy [67, 68]. Advanced neural networks and embedding techniques enable the extraction of high-dimensional features, capturing complex details and making it possible to differentiate even highly realistic AI-generated images from authentic ones [69].

Cosine Similarity

Cosine similarity is a popular metric used to quantify the similarity between two vectors by measuring the cosine of the angle between them. In image embeddings, cosine similarity determines how similar two images are based on the angle between their respective embedding vectors. This measure is particularly useful because it is invariant to the magnitude of the vectors, focusing solely on their orientation. This property makes cosine similarity well-suited for comparing image embeddings, where the direction of the vector encapsulates the semantic content of the image [70].

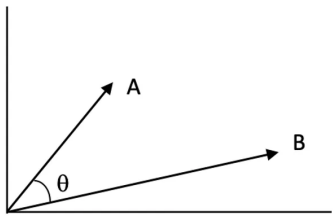


Figure 2.11: Angle between two 2-D vectors A and B

The formula for cosine similarity is given by the dot product of the two vectors divided by the product of their magnitudes. In mathematical terms, for two vectors $\mathbf{A}$ and $\mathbf{B}$ as shown in Figure 2.11, the cosine similarity is calculated as:

$$\text{Cosine Similarity} = 1 - \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\| \|\mathbf{B}\|} = 1 - \frac{\sum_i A_i B_i}{\sqrt{\sum_i A_i^2} \sqrt{\sum_i B_i^2}}$$

This metric ranges from -1 to 1, where 1 indicates identical vectors, 0 indicates orthogonal vectors, and -1 indicates opposite vectors. In practical applications, cosine similarity compares the embeddings of query images with those in a database, facilitating tasks like image retrieval and anomaly detection. Its effectiveness in capturing semantic similarity makes it a vital tool for detecting AI-generated images.

2.4.4 Vector Database

A vector database is a specialized database designed to store and manage high-dimensional vector mathematical representations of features or attributes generated by various machine learning (ML) models, particularly embedding models. These models include natural language processing (NLP) models like ChatGPT and models used for computer vision tasks.

Vector databases excel in tasks involving similarity search and classification by efficiently handling the complex, high-dimensional data representations essential for AI applications. Traditional databases, such as SQL for structured data and NoSQL for unstructured data, fall short when managing and querying this type of data due to their inherent limitations in scalability and performance.

Machine learning has transformed unstructured data into vector representations that capture meaningful relationships within the data. These vector representations, called embeddings, are used for data analysis and power many machine-learning applications. For instance, recommender systems commonly use vector embedding techniques like item2vec, word2vec, doc2vec, and graph2vec to convert items into vectors of numeric features [71]. Recommendations are then generated by identifying the items with the most similar vector representations. Similarly, images and natural language data also have inherent vector-based representations due to their numeric pixel and word components [72].

Vector databases originate from vector similarity search, where early systems were capable of similarity queries but lacked performance at scale with dynamic vector data [73, 74]. The first solutions for similarity search were either algorithms or systems. Algorithms, such as FAISS from Facebook [75], handle large volumes of data poorly, assuming all data and indexes fit into the main memory. Systems like Alibaba AnalyticDB-V are not well-suited for vector data and do not treat vectors as first-class data types. Given these issues, purpose-built vector database solutions such as Milvus [71] emerged. Milvus is a vector data management system built on top of FAISS that overcomes the shortcomings of previous solutions. It is designed specifically for large-scale vector data and treats vectors as a native data type, providing efficient and scalable management of vector embeddings.

2.5 Blockchain Technology

Blockchain is a decentralized, distributed ledger technology that allows for the secure and transparent recording of transactions across a network of computers. Each block in the blockchain contains a list of transactions and is cryptographically linked to the previous block, ensuring immutability and resistance to tampering [76]. This decentralized nature eliminates the need for intermediaries, fostering trust in peer-to-peer environments. The Figure 2.12 represents a general structure of blockchain.

2.5.1 Ethereum Network

Ethereum [77] is a decentralized open-source blockchain platform that pioneered the concept of smart contracts: self-executing code that runs directly on the blockchain. Launched in 2015 by Vitalik Buterin and others, Ethereum differentiates itself from Bitcoin by serving not just as a digital currency but as a versatile platform for decentralized applications (DApps) and programmable transactions. At its core is the Ethereum Virtual Machine (EVM), which executes scripts using a global network of public nodes. To address environmental concerns

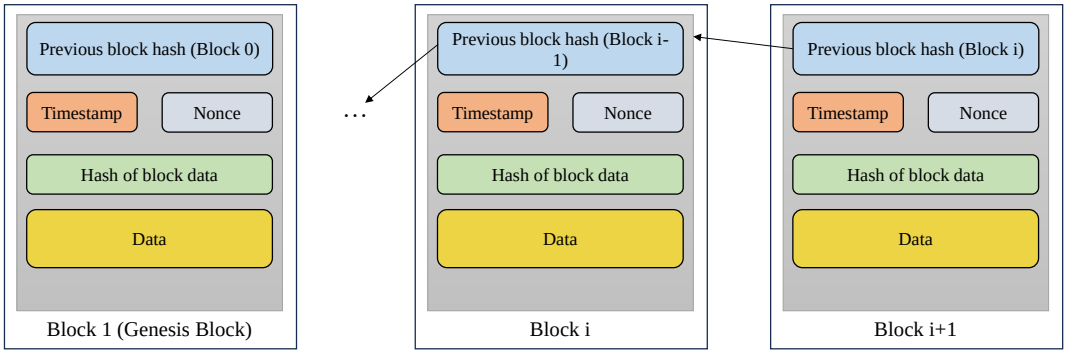


Figure 2.12: General structure of blockchain

and enhance scalability, Ethereum has transitioned from a Proof of Work (PoW) to a Proof of Stake (PoS) consensus mechanism under Ethereum 2.0.

Sepolia is a lightweight Ethereum testnet designed for developers to test and deploy smart contracts in a safe, cost-effective environment before moving to the mainnet. Unlike other testnets such as Ropsten or Goerli, Sepolia uses a proof-of-stake (PoS) consensus mechanism, aligning it with Ethereum's current mainnet protocol after the merge. Its deterministic block production, minimal resource requirements, and active support make it ideal for early-stage development and integration testing. The stability and similarity of Sepolia to the mainnet enable reliable testing of applications under near-production conditions without the risks associated with real asset transfers.

2.5.2 Smart Contracts

Smart contracts are self-executing programs stored on blockchain technology that automatically enforce the terms of an agreement when predetermined conditions are met. Originally conceptualized by Nick Szabo in 1994 [78] as "computerized transaction protocols that execute the terms of an agreement," smart contracts remained largely theoretical until blockchain technology made practical implementation possible. Szabo famously compared smart contracts with vending machines, autonomous systems that deliver products when specific conditions (inserting money) are met, without requiring intermediaries. Ethereum is the first and most popular blockchain platform to widely support smart contract functionality after Bitcoin's introduction demonstrated the feasibility of decentralized systems.

Smart contracts offer the following benefits.

- Once implemented on the blockchain, smart contracts cannot be altered, ensuring the integrity of the agreement terms.
- All contract execution events are recorded on the blockchain, making them fully traceable, reducing the risk of fraud or manipulation.

- By automating the execution of predetermined conditions, smart contracts eliminate the need for third-party intermediaries, reducing associated investigation and mediation costs.
- Without intermediaries, the contract terms are executed automatically when conditions are met, resulting in faster processing and more efficient transaction completion.

2.5.3 Programming Languages

Several programming languages have been developed specifically for blockchain and smart contract development. Languages such as Solidity, Move, and Motoko are purpose-built for blockchain ecosystems and are tailored to specific networks. In addition, several general-purpose programming languages like Rust, Go, and C++ have gained significant popularity within the blockchain development community due to their performance and versatility.

Solidity is the most widely used language for writing smart contracts, particularly on the Ethereum platform. It is a high-level, object-oriented language designed for the Ethereum Virtual Machine (EVM) [79]. With syntax closely resembling JavaScript and TypeScript, Solidity is approachable for developers with web development experience. Its accessibility, precision, and flexibility have made it a favored choice among blockchain developers.

The sample code below, written in the Solidity programming language, shows an example of a Smart contract.

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

contract HashStorageAI {
    // Dynamic array to store hashes as uint256
    uint256[] private ai_hash_list;

    // Event to emit when a new hash is stored
    event HashStored(uint256 hash, uint256 hash_list_len);

    // Function to store a single hash
    function storeHash(uint256 _hash) public {
        ai_hash_list.push(_hash);
        emit HashStored(_hash, ai_hash_list.length);
    }

    function hashExists(uint256 _hash) public view returns (bool) {
        for (uint256 i = 0; i < ai_hash_list.length; i++) {
            if (ai_hash_list[i] == _hash) {
                return true;
            }
        }
        return false;
```

```
24      }
25  }
```

2.5.4 Gas Fees

Gas fees refer to the cost of performing transactions or executing smart contracts on the Ethereum network. Because every transaction on the Ethereum network consumes computational resources, a fee is required to prevent network abuse, such as spam or infinite execution loops. This fee system ensures the network remains efficient and secure. The gas fee is calculated by multiplying the amount of gas consumed by the transaction with the current price per unit of gas. Importantly, this fee must be paid whether the transaction is successful or not [80].

These fees are paid in Ether (ETH) and are determined by the complexity of the operation and current network congestion. The native cryptocurrency of the Ethereum network is known as ether (ETH) [77]. Since transaction fees are typically a small portion of ETH, these costs are often expressed in smaller denominations to simplify calculations and improve readability. **Wei** is the smallest unit of ETH, where **1 ETH** $= 10^{18}$ **wei**. Another commonly used denomination is **Gwei**, which equals 10^9 **wei** [80]. Gwei serves as a convenient middle-ground unit, making it ideal for representing gas fees—suitable for both small and large values. Other blockchain platforms, such as **Avalanche** and **Solana**, use their own native cryptocurrencies and unit systems for similar purposes.

2.5.5 Decentralized Applications (DApps)

DApps are software applications that run on a decentralized blockchain network, utilizing smart contracts to execute logic on-chain. In practice, DApps are universally available web services running on the EVM. Users can, however, access the DApp via a web browser or smartphone application. They often interact with blockchain through wallets like MetaMask and use front-end technologies such as React or Vue.js.

2.5.6 Crypto Wallets

Cryptocurrency wallets are software tools that allow users to securely store, manage, and interact with their digital assets such as cryptocurrencies and tokens [81]. These wallets serve as an interface between users and blockchain networks, enabling them to send, receive, and store assets. However, unlike traditional wallets that hold physical currency, crypto wallets do not store the actual cryptocurrency. Instead, they store the private and public keys that are required to access and manage assets on the blockchain. The assets themselves reside on the blockchain, and ownership is proven through the private key. This key functions like a secure digital password, authorizing transactions and access. Losing the private key means losing access to the corresponding digital assets—there is no way to recover the funds without it. For this reason, securely storing private keys is crucial. Saving them in unsecured

locations such as internet-connected devices or plain text files exposes them to threats like hacking, malware, and unauthorized access.

To mitigate these risks, crypto wallets implement various security features including encryption, multi-factor authentication, and cold storage. When private keys are stored offline, typically in a hardware device, the wallet is known as a cold wallet [81]. Cold wallets are not connected to the internet, making them highly secure but less convenient for frequent transactions. In contrast, hot wallets store keys on devices with internet access, offering greater ease of use and speed but with increased vulnerability to security breaches.

MetaMask

MetaMask is a popular hot wallet that facilitates seamless interaction with the Ethereum network and various blockchain applications [82]. Available as both a browser extension and a desktop application, MetaMask allows users to securely manage Ethereum-based tokens, execute transactions, and interface with smart contracts.

Users can also switch effortlessly between different Ethereum networks—including the mainnet, testnets, and custom private chains—making it highly flexible. MetaMask's user-friendly interface, strong security practices, and compatibility with numerous decentralized applications (dApps) have made it a preferred choice among Ethereum users.

2.5.7 Limitations of Blockchain Technology

Blockchain technology has made advancement across a wide range of sectors. Despite its growing influence, blockchain is not without limitations. Like any emerging technology, it faces several challenges related to implementation, operation, and maintenance that can hinder its overall effectiveness [83].

- **Scalability:** Blockchain networks, by design, require each node to verify and store transactions to maintain consensus and security. This distributed nature, while beneficial for decentralization, significantly impacts performance. Every transaction must go through a consensus mechanism, which involves complex verification and redundancy. Unlike centralized systems that can process requests in parallel, blockchain nodes must process transactions sequentially, which slows down the system. Moreover, as the blockchain grows, storage demands increase rapidly, posing challenges for long-term scalability.

- **Energy Usage:** Blockchain networks, particularly those using Proof of Work (PoW) consensus mechanisms, demand substantial computational resources. Mining new blocks involves solving complex cryptographic puzzles, which requires high-performance hardware and consumes significant amounts of electricity. As blockchain adoption increases and networks expand, the associated energy consumption rises dramatically. This makes blockchain a less sustainable technology compared to traditional systems, especially from an environmental perspective.

- **Privacy:** While blockchain provides transparency and immutability, it lacks built-in privacy protections. Every transaction is visible on the public ledger, which can expose user behavior patterns and sensitive data if identities are linked to addresses. Furthermore, errors in data entry or human mistakes can lead to permanently stored inaccurate or compromising information. Since there is no central authority to validate or redact this data, maintaining privacy remains a critical issue.

- **Regulation and Compliance:** Blockchain's decentralized and pseudonymous nature complicates regulatory oversight. There are currently no universal standards or frameworks for blockchain governance, which creates uncertainty for organizations and users. Additionally, the irreversible and immutable nature of blockchain data can pose compliance risks—especially with regulations like GDPR, which require mechanisms for data correction or erasure. The absence of consistent global regulations hinders mainstream adoption and raises legal and ethical concerns.

2.6 Related Work

Diffusion models continue to advance the field of image generation, setting new benchmarks for quality and efficiency. One of the notable developments is the introduction of the Stable Cascade [84] model by Stability AI, which represents a major leap forward from its predecessor, Stable Diffusion XL (SDXL). Stable Cascade offers faster performance, increased cost-efficiency, and enhanced user-friendliness, making it a compelling choice for text-to-image diffusion tasks. This model achieves these improvements through optimized noise scheduling and advanced architectural designs that streamline the generation process, reducing computational demands while maintaining high image fidelity [85, 86]. Enhanced conditional generation capabilities have also emerged, with methods such as classifier-guided diffusion and classifier-free guidance [87], making these models more versatile and effective in producing images based on specific conditions. Further advancements include the development of pipelines using multiple diffusion models at increasing resolutions to generate high-resolution images, coupled with noise conditioning augmentation to maintain high fidelity [88]. Innovative architectural changes introduced by ControlNet [89] allow for conditioning on additional images, enabling the generation of images based on complex inputs like Canny edges and human pose skeletons. Lastly, the Diffusion Transformer (DiT) model [90], which operates on latent patches, leverages the design space of Latent Diffusion Models (LDM) to enhance both the quality and efficiency of image generation through the use of transformers.

Table 2.1: Recently published research on identifying AI-generated images

Published Paper	Main Ideas	Dataset Used	Pros	Cons	Method Used
On the detection of synthetic images generated by diffusion models [91]	Assessed the difficulty of distinguishing diffusion model-generated synthetic images from real ones and evaluated the effectiveness of current detectors for GAN-generated images. Analyzed forensic traces and tested deep learning detectors under conditions like image compression and resizing typical of social media.	COCO (Common Objects in Context), ImageNet, UCID	Provides insights into forensic traces left by diffusion models, crucial for developing effective detection methods. Compares detector performance, highlighting strengths and weaknesses in identifying diffusion model-generated images. Analyzes scenarios, including social media-like conditions, for practical relevance.	Detector performance is inconsistent across different models, indicating a lack of generalization in current detection methods. Reliance on forensic traces that can be easily altered by common image processing techniques like compression and resizing limits the effectiveness of these detectors in practical applications.	Deep Learning (DL)
Towards Universal Fake Image Detectors That Generalize Across Generative Models [92]	Used CLIP's feature space for real-vs-fake classification via nearest neighbor and linear probing.	ImageNet	Improves a standard image classifier trained on ProGAN by +15.07 mAP and +25.90% acc on unseen diffusion and autoregressive models.	Relies on feature space from a pretrained model, which is computationally intensive and specific to that model.	DL
Unmasking Deception: Empowering Deepfake Detection with Vision Transformer Network [93]	Uses a pre-trained ViT model fine-tuned for deepfake detection. Treats images as sequences of patches, utilizing self-attention to capture local and global features. Processes embedded fixed-size patches through transformer layers.	140 k Real and Fake Faces	The fine-tuned ViT model achieves near-perfect metrics. Explainable AI techniques enhance transparency, revealing features used for classification.	ViTs require significant resources for training and inference. Performance may degrade with novel or sophisticated deepfake techniques not included in training data.	Vision Transformers (ViTs)
GLFF: Global and Local Feature Fusion for AI-Synthesized Image Detection [94]	Used CNN-based feature extraction, an attention-based multi-scale feature fusion (AMSFF) module, and a Patch Selection Module (PSM) to aggregate features from ResNet-50. Features are fused with an attention mechanism and fed into a binary classifier to distinguish real from AI-synthesized images.	DeepFake-Face-Forensics (DF3)	Performs well on seen and unseen data, including images with post-processing like JPEG compression, Gaussian blur, and face blending.	Reliance on CNN-based feature extraction and attention mechanisms is computationally intensive. Performance may degrade under extreme post-processing like anti-forensics.	DL
AI vs. AI: Can AI Detect AI-Generated Images? [95]	Used a pre-trained EfficientNetB4 model, fine-tuned on a new dataset of real and GAN-generated images. The dataset included various synthesis models. Applied transfer learning and integrated Class Activation Maps (CAM) to identify discriminative regions, providing an explainable AI approach.	Real or Synthetic Images (RSI)	Achieved 100% accuracy on the RSI dataset, with excellent generalization across datasets and modalities. Using GradCAM, AblationCAM, LayerCAM, and Faster ScoreCAM enhanced transparency and reliability by providing insights into the model's decisions.	Performance relies on the quality and diversity of the training dataset. Biases or gaps can affect effectiveness. Training EfficientNetB4 and generating CAMs are computationally intensive and challenging under extreme conditions.	DL
Detection of AI-Created Images Using Pixel-Wise Feature Extraction and Convolutional Neural Networks [96]	Combined pixel-wise feature extraction methods and CNNs for binary image classification. The two techniques are: i) PRNU, a unique noise pattern from camera sensors, differentiates real from AI-generated images; ii) ELA detects inconsistencies in compression errors, highlighting manipulation or AI generation.	Dresden Image Database, VISION Dataset	The study achieves over 95% accuracy using PRNU and ELA features. CNNs enhance detection by learning subtle differences. GradCAM provides insights into specific features and regions contributing to the classification, increasing transparency and reliability.	Dependence on PRNU and ELA requires JPEG images, limiting applicability to other formats. Training CNNs and generating PRNU and ELA patterns are resource-intensive. Performance may degrade under extreme manipulations or adversarial attacks designed to evade detection.	DL
AI-Generated Image Detection using a Cross-Attention Enhanced Dual-Stream Network [97]	Introduces a dual-stream network: i) Residual stream captures high-frequency texture anomalies using a Spatial Rich Model, ii) Content stream captures low-frequency forged traces. Features are integrated with a cross-multi-head attention mechanism, then passed through CNN layers and a classifier to determine if an image is AI-generated or real.	ALASKA Database, DsTok Dataset, SPL2018 Dataset	The authors constructed DALL·E2 and DreamStudio databases to evaluate their method, which outperformed comparable methods on CG detection datasets DsTok and SPL2018.	The dual-stream architecture with cross-attention requires substantial computational resources. Effectiveness depends on SRM and content stream features, which may be less robust against new generative models producing different artifacts.	DL
The Stable Signature: Rooting Watermarks in Latent Diffusion Models [98]	Embed invisible watermarks in AI-generated images using Latent Diffusion Models (LDMs). The method involves two phases: i) Pre-training a watermark extractor with a simplified HiDDeN method, ii) Fine-tuning the LDM's decoder to embed a fixed binary signature into generated images.	MSCOCO Validation Set	Embeds watermarks without altering image generation, making it efficient and secure. Robust against transformations, maintaining high accuracy even with cropping or compression. Allows detection and identification of AI-generated images, aiding in tracking and verification.	Effectiveness depends on the pre-trained extractor and fine-tuning quality. Watermarks may be less detectable with extreme tampering. Fine-tuning requires access to generative models, which may not be feasible for closed-source models, and sophisticated attacks could evade detection.	Watermarking

Published Paper	Main Ideas	Dataset Used	Pros	Cons	Method Used
CIFAKE: Image Classification and Explainable Identification of AI-Generated Synthetic Images [99]	Used a CNN to classify images as real or AI-generated, comparing synthetic images from Stable Diffusion 1.4 with real images from CIFAR-10 in the CIFAKE dataset. Fine-tuned the model through hyperparameter optimization and used Grad-CAM to visualize classification features.	CIFAR-10, CIFAKE	Achieved 92.98% accuracy in distinguishing real from AI-generated images using the CIFAKE dataset, combining CIFAR-10 and LDM-generated images.	Generalizability to other AI-generated images or models remains untested. Future generative models with fewer imperfections may challenge the method's reliance on subtle artifacts.	DL
Multiclass AI-Generated Deepfake Face Detection Using Patch-Wise Deep Learning Model [100]	Used Vision Transformers (ViTs) to detect deepfake images by extracting global features. The model divides input images into non-overlapping patches to capture global and local information, contrasting with traditional CNN models focusing on local features.	140k Real and Fake Faces, Synthetic Faces High Quality (SFHQ)	Outperformed ResNet-50 and VGG-16 in detecting deepfake images, achieving a 99.90% F1 score. Introduced a multiclass approach, addressing challenges with Stable Diffusion and StyleGAN2.	Vision Transformers require significant computational resources. Dependence on specific training datasets may limit effectiveness against novel manipulation techniques not covered in the data.	ViTs
A Single Simple Patch is All You Need for AI-generated Image Detection [101]	The SSP network distinguishes real from fake images by leveraging noise patterns. It identifies the simplest patch with the least texture diversity, processes it with SRM filters to extract high-frequency noise patterns, and uses a fine-tuned ResNet-50 classifier for detection.	GenImage Dataset, ForenSynths Dataset	The SSP network generalizes well across generative models and image degradations, outperforming existing methods on GenImage and ForenSynths datasets.	The SSP network's performance degrades with image blurring or compression, showing sensitivity to image quality.	DL
GenImage: A Million-Scale Benchmark for Detecting AI-Generated Image [102]	Analyzed the GenImage dataset using two tasks: i) Cross-Generator Image Classification tests detectors' generalization by training on one generator's images and testing on others. ii) Degraded Image Classification assesses robustness against degradations like low resolution, JPEG compression, and Gaussian blur.	GenImage Dataset	Introduces the GenImage dataset for comprehensive evaluation of AI-generated image detectors, simulating real-world scenarios with unknown models. Uses frequency analysis and generator correlation to highlight detection challenges of AI-generated images.	The reliance on ImageNet for real images inherits its biases, affecting result generalizability. Generating and processing a large dataset is computationally demanding. Although it includes various image classes, it may not fully capture real-world image diversity.	DL, ViTs
MaskSim: Detection of Synthetic Images by Masked Spectrum Similarity Analysis [103]	Used a semi-white-box method to detect synthetic images by analyzing masked spectrum similarity, leveraging the Fourier spectrum to identify generative model patterns and training a mask to amplify discriminative frequencies and a reference pattern to enhance detection accuracy.	Synthbuster, PolarDiffShield, Mit-5k, Raise, HDR-Burst, Dresden, COCO	The method achieves high detection accuracy, provides explainable results, and generalizes well across generative models. It remains robust even after common post-processing operations like JPEG and WebP compression.	The method requires significant computational resources and relies on the quality and diversity of the training dataset. Performance degrades with rescaled or heavily manipulated images and is limited to detecting entirely synthetic images.	DL

Chapter 3

EmbedAIDetect System Design

This chapter presents an overview of the design and architecture of EmbedAIDetect, a system developed for the reliable detection of AI-generated images. EmbedAIDetect integrates two core components: (1) an embedding-based similarity analysis module for detecting synthetic content, and (2) a blockchain-backed verification mechanism to ensure the integrity and provenance of embeddings.

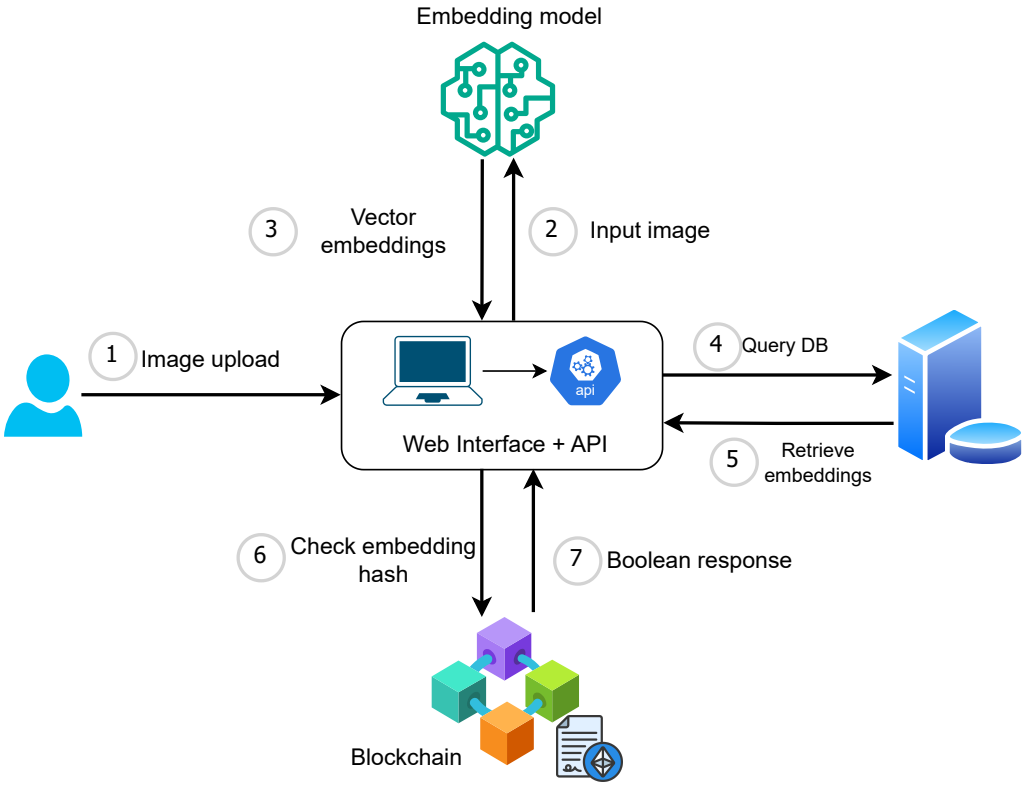


Figure 3.1: System diagram of EmbedAIDetect

The final system design is the outcome of an iterative development process involving three successive prototypes. Each prototype progressively introduced key architectural concepts that shaped the system as it stands. The first prototype implemented a blockchain-only

approach, storing cryptographic hashes of image embeddings for on-chain verification. The second prototype shifted toward a vector similarity-based method using a vector database, enabling more flexible and semantically aware image classification. The third and final prototype integrated both strategies into a hybrid model that combines semantic similarity with verifiable provenance. These evolving prototypes formed the basis for evaluating the design and influenced the architecture of EmbedAIDetect, which merges both components into a cohesive detection and verification framework.

The EmbedAIDetect AI-generated image detection framework is designed to integrate a vector database with the embedding model and a user-friendly interface. The framework implements a step-by-step pipeline for analyzing whether an unseen (uploaded) image might be AI-generated. Figure 3.1 illustrates the architecture design for EmbedAIDetect. It provides a user-friendly interface to upload an image and check the results. The results are based on the embedding vectors that are stored in the vector database. The following sections provide a detailed description of each component and their interactions, key modules, data flow, and the rationale behind critical design choices.

3.1 Embedding Model

An image embedding model transforms images into fixed-length vector representations that capture their semantic content. These dense embeddings position semantically similar images close together in a high-dimensional vector space, enabling efficient similarity comparison. In the context of AI-generated image detection, the quality and consistency of these embeddings are critical.

After evaluating five different image embedding models on how their vector representations respond to common image manipulations, DINOv2 [104], a self-supervised Vision Transformer (ViT) model emerged as the most suitable for the EmbedAIDetect system. Smilar to a BERT-like transformer encoder, it treats each image as a sequence of fixed-size patches and produces a compact representation using the final hidden state of a special classification token. In the system workflow, input images are passed through the DINOv2 embedding function, which processes the image and extracts a normalized [CLS] token as its embedding, then used for similarity comparison.

3.2 Vector Database

Vector databases are specialized systems for efficiently storing and querying high-dimensional vector data. Unlike traditional SQL databases, which focus on storing and retrieving structured data, they lack the tools to efficiently index and compare vectors based on similarity metrics such as Euclidean distance or cosine similarity. These databases implement specialized indexing structures, such as HNSW (Hierarchical Navigable Small World) or IVF (Inverted File Index) [105], which enable efficient approximate nearest-neighbor search, crucial for finding similar embeddings quickly.

The EmbedAIDetect system employs *ChromaDB* [1] as the vector database solution to manage image embeddings. ChromaDB is an open source, lightweight, and serverless platform that makes it easy to set up and run without managing a separate back-end service. It supports persistent local storage, and collections can be created or retrieved programmatically using a simple Python API. It supports integration with popular embedding providers such as OpenAI and Hugging Face. It allows embedding functions to be defined at the time of collection creation, enabling automatic embedding generation during data ingestion or optional manual control when needed [106].

ChromaDB also provides flexible features for querying, including efficient similarity search and metadata filtering, allowing retrieval of embeddings based not only on vector closeness but also on additional image attributes. In the system workflow, when a new image is added, its embedding is generated using the DINOv2 model, and the image metadata and embedding are added to the appropriate collection in ChromaDB. This enables rapid and scalable similarity comparisons during detection.

3.3 Blockchain

Blockchain is a decentralized digital ledger that ensures data integrity and transparency without relying on a central authority. Data are stored in cryptographically linked blocks, which makes it highly tamper resistant. Among blockchain platforms, Ethereum stands out for its support of smart contracts, which are self-executing programs that run on-chain. This capability allows for the building of decentralized applications (dApps) that automate and verify processes in a trustless environment. Although newer platforms such as Solana and Avalanche offer faster transactions and lower fees [107], Ethereum remains widely adopted due to its robust security, mature ecosystem and strong developer support.

For this thesis project, the Sepolia test network, an Ethereum-based testnet, is chosen to implement and prototype the blockchain-backed verification module. Sepolia offers the advantages of Ethereum's architecture while eliminating transaction costs, as test Ether (ETH) can be acquired freely through public faucets [2], typically providing up to 0.05 ETH per day. It mirrors Ethereum's behavior, making it ideal for testing smart contracts under realistic conditions without financial risk. Sepolia's stability, active support, and ease of integration made it a practical and reliable choice for validating the design and functionality of the EmbedAIDetect system during development.

3.4 Web App

The web application serves as the central layer in the EmbedAIDetect system, integrating the embedding model, vector database, and blockchain verification components into a cohesive

[1] `https://docs.trychroma.com/docs/overview/introduction`

[2] `https://cloud.google.com/application/web3/faucet/ethereum/sepolia`

pipeline. Its primary role is to manage user interaction, coordinate data flow between system components, and consolidate results for presentation.

The system follows a client-server architecture in which the web interface allows users to upload an image for analysis. Upon upload, the image is processed by the embedding model, which generates a high-dimensional vector representation. This vector embedding is then used to perform a similarity query against the vector database to identify whether the input closely matches existing embeddings of AI-generated or human-generated images.

At the same time, the web application computes a hash of the embedding and initiates a verification step through the blockchain module. This hash is checked against a smart contract ledger to determine whether it matches a previously registered embedding. The result of this verification, indicating whether the embedding is recognized and verifiable, is combined with the similarity results from the vector database to form a final response. This entire workflow is managed through internal API calls and follows the sequence illustrated in the system design diagram 3.1.

3.5 User Story

Figure 3.2 shows the overall interaction between a user and components of the EmbedAIDetect AI-generate image detection system. The detection process is initiated by uploading an image (suspected to be AI-generated or otherwise) through a Web Interface. This interface serves as the front-end component. Upon receiving the image, the system forwards it to the embedding model, which generates a high-dimensional embedding vector.

Following the embedding generation, the system enters its analysis phase. The embedding vector is taken as input by the back-end script to query the vector database. The query is done with cosine similarity as a metric to find the closest match with the existing data in the database. The query results include the closest match embedding and its metadata. The metadata has the filename, hash, and location of the image file in local storage so that it can show to the user what the input image matches, along with similarity scores. After analyzing the database query results, which include the closest distance to both AI training data and human training data, the system computes the likelihood estimate based on the query results, which are AI-generated.

The system incorporates a crucial feedback loop for continuous improvement. After displaying the analysis results to the user, an optional validation step allows users to confirm or correct the system detection. If the user already knows that the input image is real or fake, this validation step will increase the system's accuracy, since human validation provides valuable ground-truth data. When users provide feedback, the system stores this validated information through the back-end script and persists it in the database, creating a growing corpus of verified examples. The storage of this feedback is confirmed through the system and any resulting improvements in the detection mechanism are communicated back to the user.

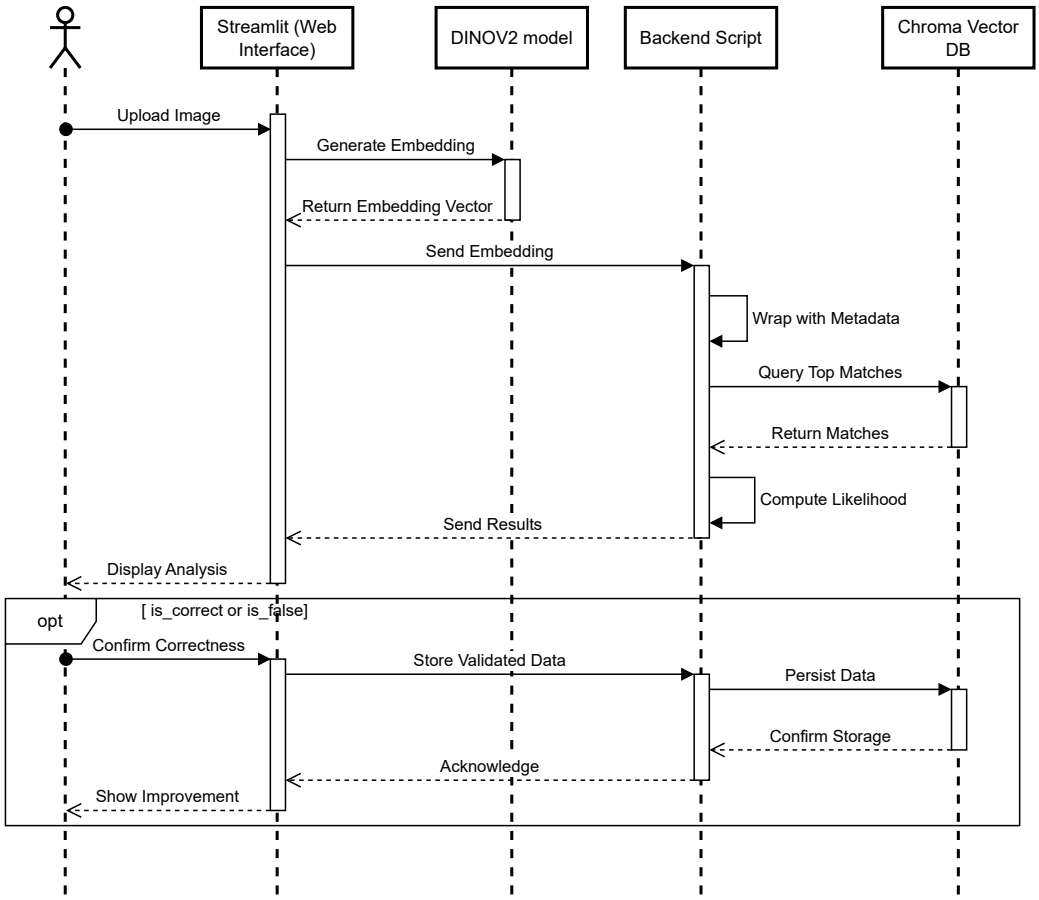


Figure 3.2: Sequence diagram of the EmbedAIDetect system

Chapter 4

Prototype Implementation

In the previous chapter, we discussed the system design for EmbedAIDetect. This chapter provides the implementation details for the EmbedAIDetect prototype developed incrementally through three distinct prototype versions, where each version builds on the previous one. The goal of such a development process is to show how each prototype was conceptualized, implemented, and improved to build the final system. Each version introduces new components—starting from a blockchain-only setup, to incorporating a vector database for similarity search, and finally combining both for verification and classification. Detailed flowcharts and explanations for each prototype are provided in the following sections to illustrate the application logic and architectural decisions.

4.1 Prototype 1: Blockchain only

The application setup as shown in Figure 4.1 for the first prototype focuses on the preparation of the blockchain environment and the datasets required for the image classification process. The system begins by initializing two arrays: one for AI-generated images and one for real (human) images. A total of 7,000 images from each category are loaded. For every image in the datasets, the system extracts embeddings using the DINOv2 model. These high-dimensional embeddings are then converted into 256-bit hashes. The resulting hashes from AI-generated images are stored in a smart contract called HashStorageAI, whereas the hashes from real images are stored in a separate contract named HashStorageHuman. This structure ensures a clear and immutable distinction between the two types of image origins, entirely on-chain, without relying on any external database.

Figure 4.1 is a flow chart for the use of the prototype after setup. When a user uploads an image for classification, the application extracts its embedding using the same DINOv2 model and computes the corresponding 256-bit hash. The system then checks this hash against the two smart contracts: HashStorageAI and HashStorageHuman. If a match is found in HashStorageAI, the image is classified as AI-generated. If the hash exists in HashStorageHuman, it is labeled as a real image. In cases where the hash is not found in either contract, the result remains inconclusive. This approach leverages the immutability and transparency of blockchain to verify the authenticity of the image without involving a traditional database or external storage mechanism.

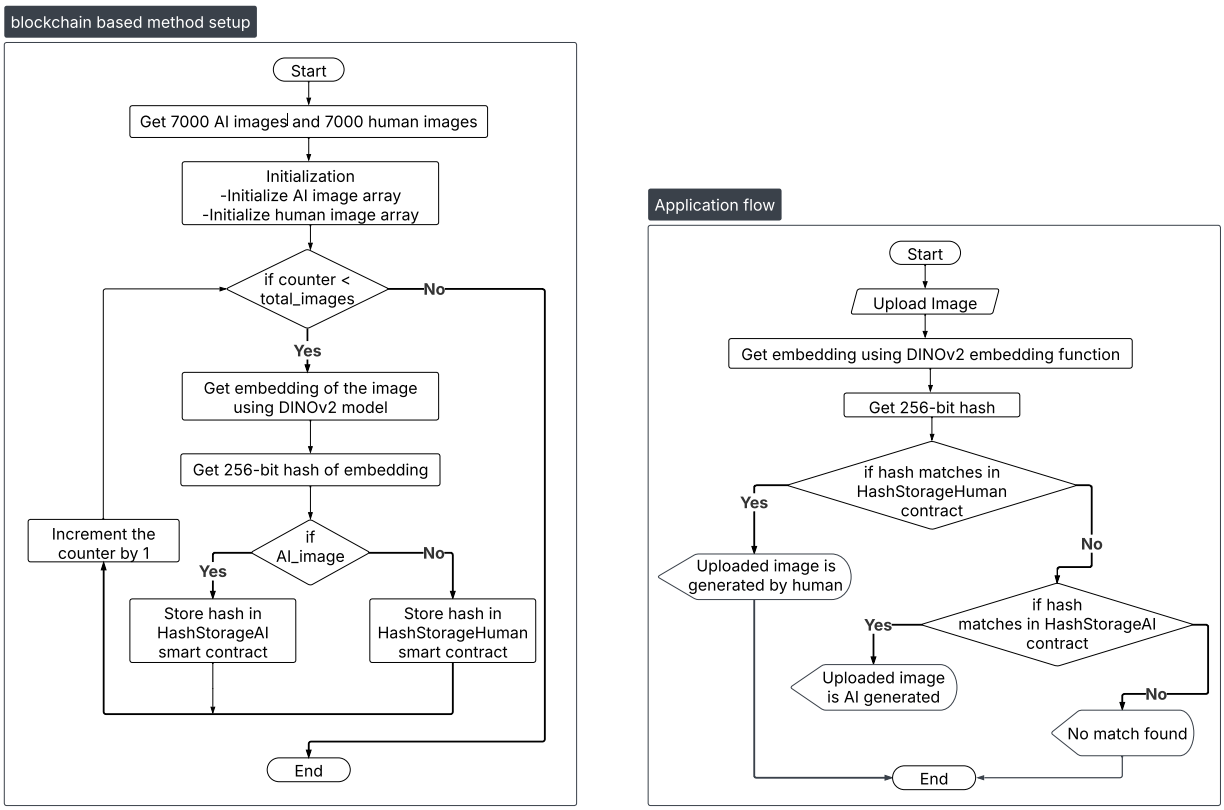


Figure 4.1: Application setup and data flowchart illustrating the image upload process and hash-based classification using smart contracts in the blockchain-only prototype of EmbedAIDetect

4.2 Prototype 2: Database only

The application setup as shown in Figure 4.2, is the second prototype which introduces a vector similarity-based approach using a vector database, specifically ChromaDB. The setup begins by initializing two distinct collections: one for AI-generated images and one for real (human) images. Each collection is populated with 7,000 images, respectively. For every image, the DINOv2 image embedding model is used to extract high-dimensional vector embeddings. These embeddings are then stored in the appropriate collection along with their corresponding metadata. This configuration allows the system to leverage efficient similarity search operations for classification without involving the blockchain at this stage.

Upon uploading a test image, the system calculates its embedding using the same DINOv2 model. This embedding is then used to query both the AI and the human collections within the vector database. The system retrieves the closest match from each collection and calculates the distance to both. If the distance to the AI collection is smaller than that to the human collection, the image is classified as “Likely AI.” Otherwise, it is classified as “Likely Human.” This approach emphasizes semantic similarity over cryptographic proof,

aiming to classify images based on how closely their embedding aligns with known examples from each category.

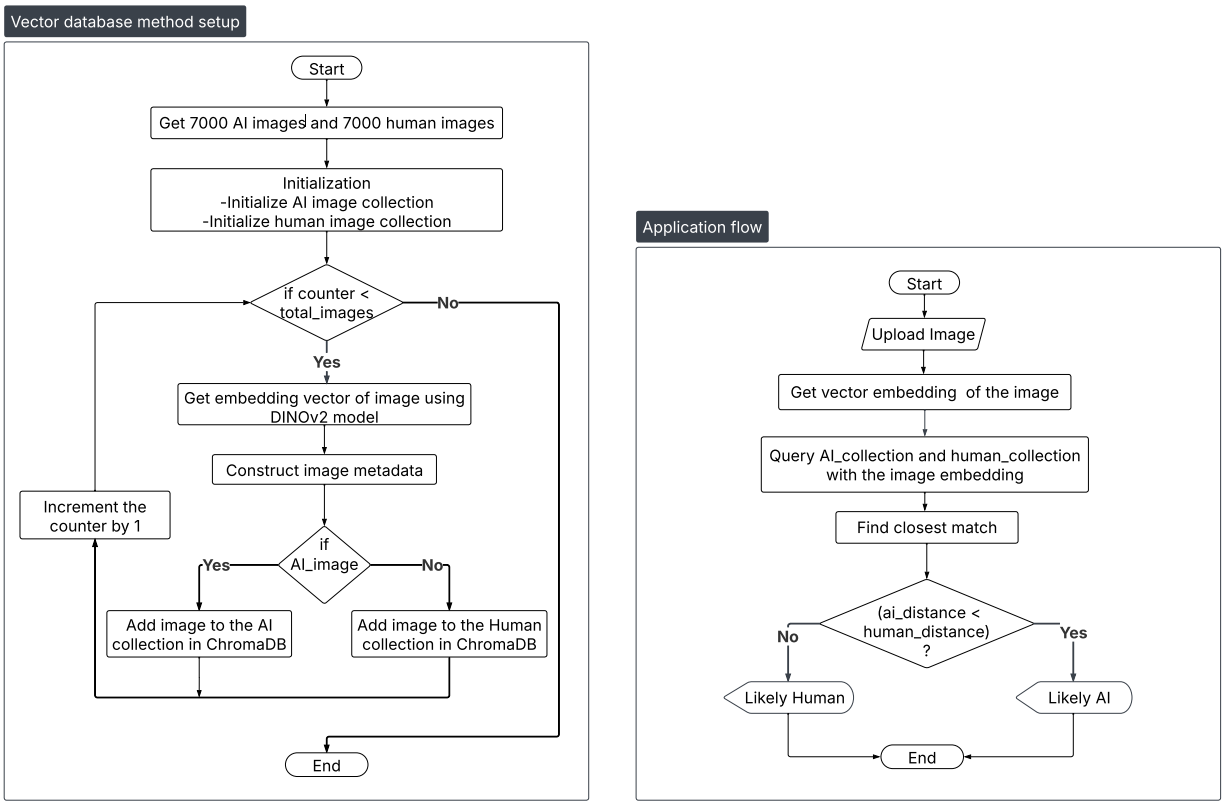


Figure 4.2: Flowchart illustrating the setup and image classification process using vector similarity search in ChromaDB for the vector database prototype of EmbedAIDetect

4.3 Prototype 3: Hybrid Approach

The third prototype flow chart, as shown in Figure 4.3 integrates both blockchain and vector database approaches to form a hybrid system. It begins by initializing two collections, `AI_collection` and `human_collection` in ChromaDB vector database, each populated with 7,000 images. For every image, the system extracts vector embeddings using the DINOv2 model, constructs the necessary metadata, and stores the embeddings in the corresponding collection. In parallel, a 256-bit hash is computed from each embedding and stored in a blockchain-based smart contract (`HashStorageAI` or `HashStorageHuman`), depending on the image's origin. The prototype architecture ensures that every embedding used for similarity search is also immutably anchored on the blockchain for later verification.

When a test image is uploaded, its embedding is calculated using the DINOv2 model and is used to query both the AI and human collections in ChromaDB. The system identifies the

closest match from each collection and compares their distances to determine to which group the image is most likely to belong to. Once the likely category is identified, the embedding of the closest matched image is retrieved, and a 256-bit hash of that embedding is computed. This hash is then verified against the blockchain through the appropriate smart contract. If the hash is found, the classification is declared valid and verifiable. If the hash does not exist on the chain, the result is considered unverified and the system may label the image as "undecided." This hybrid method ensures both semantic accuracy and tamper-proof verification.

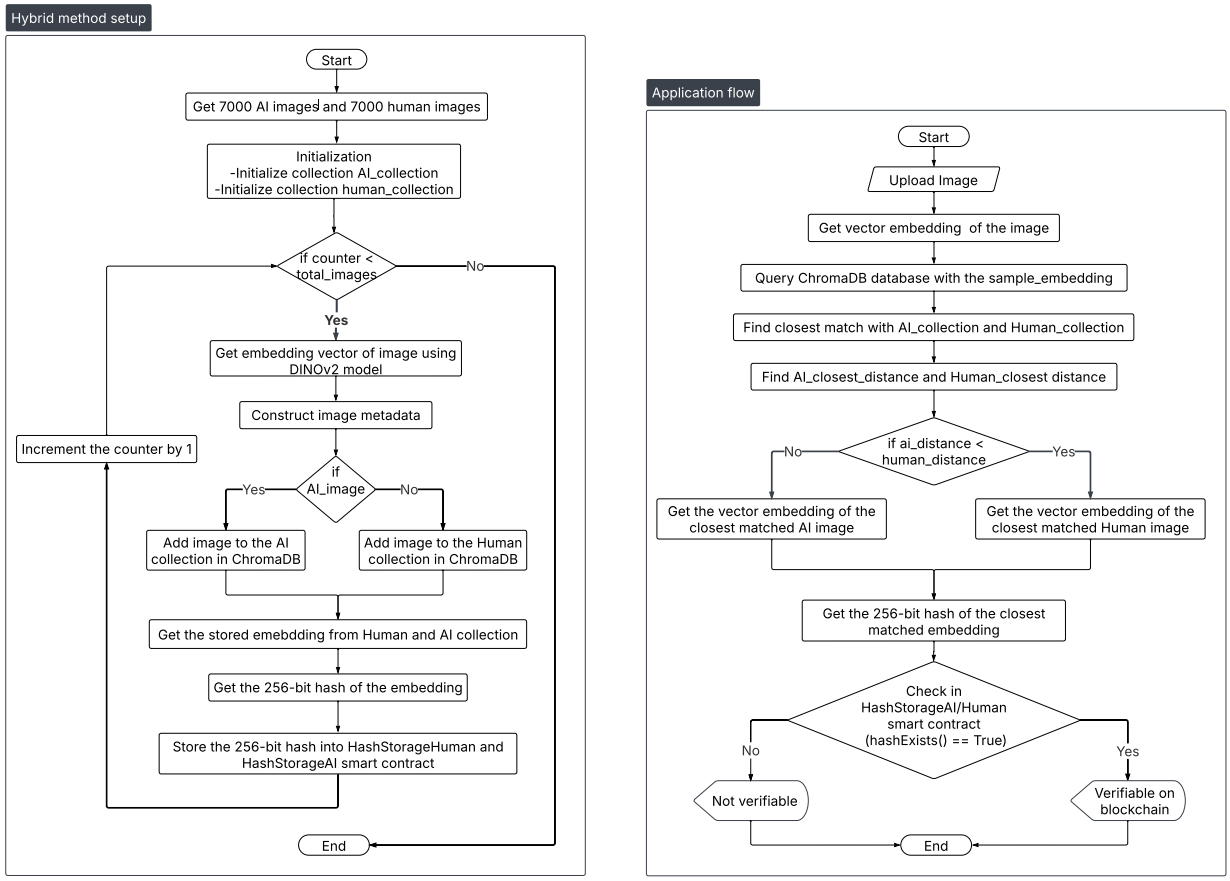


Figure 4.3: Flowchart illustrating the hybrid prototype of EmbedAIDetect, combining vector similarity search and blockchain-based verification to classify and authenticate uploaded images

Table 4.1: Tools used for prototype development.

Functionality	Chosen tool
User Interface	Streamlit v1.41.x
Programming Language	Python v3.12.x
Vector Database	ChromaDB v0.6.x
Embedding Model	Transformers v4.48.x
Smart Contract Language	Solidity v0.8.x
Smart Contract Development	Hardhat v2.23.x
Crypto Wallet	MetaMask
Web3 Provider	Alchemy's API
Blockchain Network	Ethereum Sepolia Testnet
Package & Environment Management	Conda v24.11.x

4.4 Components Development

This section outlines the main components developed during the implementation phase, detailing how each part of the system was constructed and how they collectively contribute to the functionality of the final prototype. The components mirror the architectural modules described above, ensuring traceability from design to execution. All tools used for the development of EmbedAIDetect prototype components are listed in Table 4.1.

4.4.1 Embedding Model

We created a custom embedding function for the embedding model using the *AutoModel* and *AutoProcessor* classes from the *Hugging Face* `transformers v 4.48.x` library. The processor transforms each input image into tensors that the vision transformer can interpret directly, and the entire set-up is shifted to a GPU (cuda) device for accelerated computation. When the model is called on an image file, the file is pre-processed before entering the forward pass of the neural network. During this pass, the model produces a tensor of patch-level hidden states for each image. To condense these patch representations into a single embedding, the hidden states are averaged across the patch dimension. The resulting vector is then moved to CPU memory, converted into a NumPy array, and flattened into a conventional Python list. This flattened list ensures compatibility for downstream tasks such as insertion into the vector database *Chroma DB*.

4.4.2 Embeddings Storage Setup

We used an open source vector storage library *Chroma DB* as the database for the prototype. It maintains specialized collections optimized for similarity searches in high-dimensional spaces, making it well suited for identifying semantically related images. The database is initiated as `PersistentClient` by specifying a local storage path. Any specific embedding

model like DINOV2 needs to be wrapped in a function and can be passed as a parameter when creating collections. Collections, where embeddings, documents, and any additional metadata are stored, are groups that are similar to tables in relational databases. Each collection, dedicated to AI-generated or human-art images, is configured with an embedding function and metadata specifying the distance metric (in this case, cosine). When new data arrive, the embedding function automatically transforms them into numerical vectors for storage. Chroma DB supports querying stored embeddings using both embedding-based and text-based search modes. We used querying by embeddings for the application as the user provides an unseen image for which the embedding function generates vector embedding, and the database returns the top k closest matches. The results are handled separately by a back-end Python script for analysis and displayed to the user via *Streamlit*.

4.4.3 Smart Contract Setup

To integrate blockchain functionality into the system, the Ethereum [77] network was selected as the underlying blockchain platform due to its robustness, wide adoption, and comprehensive developer tooling. Smart contracts were written using Solidity, Ethereum's native programming language, allowing for efficient development and deployment of on-chain logic. Specifically, two contracts `HashStorageAI` and `HashStorageHuman` were created to store 256-bit hash representations of image embeddings for AI-generated and human-art (real) images, respectively. The contracts were developed and tested locally using Hardhat, a popular Ethereum development environment that supports Solidity compilation, contract deployment, and local testing. To deploy the contracts in a live test environment, the Ethereum Sepolia test network was used. Sepolia was chosen for its lightweight structure, Proof-of-Stake (PoS) consensus, and compatibility with the Ethereum mainnet architecture. Deployment to the Sepolia network required access to Web3, for which the setup of an Alchemy account was required, which provided the RPC URL and API key needed to establish an authenticated connection between the Python development environment and the Ethereum network.

For the interaction between the deployed smart contracts and the backend system, a Web3 interface was implemented using Python's Web3.py library. This library enables programmatic communication with Ethereum nodes through JSON-RPC, allowing the back-end to execute read/write operations on smart contracts. To simplify this integration, a custom wrapper class was developed in Python to abstract away the complexity of raw contract calls. This wrapper handles contract initialization, function binding using the Application Binary Interface (ABI), and transaction signing with MetaMask. MetaMask was used as the crypto wallet to securely store the private keys and authorize transactions, such as adding new hashes to the blockchain. The wrapper class exposes high-level functions like `store_hash()` and `check_hash_exists()`, enabling smooth interactions with the blockchain layer from within the web application.

4.4.4 User Interface

We leveraged *Streamlit* [1] to build a simple single page user interface. It allows data scripts to be turned into web applications with minimal code and provides a reactive web application framework that manages the state of the application while handling data passing between components. This web interface acts as the integration layer that connects the embedding model, the vector database, and the blockchain verification module, forming a cohesive and interactive end-to-end system.

The interface follows a client-server architecture, where the server-side logic handles the core functionality, including embedding computation, database querying, and blockchain verification. A `file uploader` component is used to upload an image, ensuring that only valid image formats such as JPG, JPEG, PNG and WEBP are accepted. Once an image is uploaded, it is displayed within the application using Streamlit's `image` component. A `button` component then triggers a structured sequence: the image is passed to the embedding model to generate an embedding. This embedding is used to query the vector database for similarity matches. Internal API calls manage this interaction and data flow, abstracting the complexity from the user.

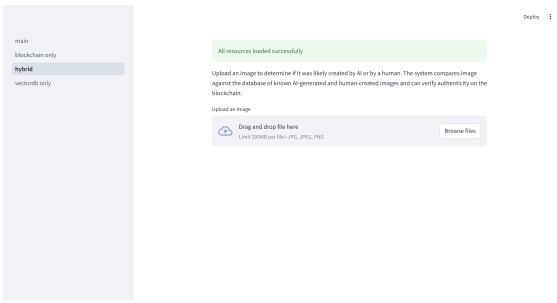
During the process, a progress bar and real-time status updates inform users about the different stages, including embedding generation, similarity search, and final classification. Detection results are displayed using `alert` components, which indicate whether the uploaded image is likely to be AI-generated or human-generated. A confidence score is also provided, derived from the cosine similarity distance between the query image and the closest matches in the AI and human image collections.

In addition to off-chain similarity scoring, the web interface incorporates blockchain-based verification to enhance the trustworthiness of the result. After generating the embedding, the Web application derives a 256-bit hash and queries the deployed smart contracts—`HashStorageAI` and `HashStorageHuman`—on the Ethereum blockchain to check if the hash already exists. Based on the response, the interface appends a verification status, such as 'verifiable' or 'not verifiable', to the classification output.

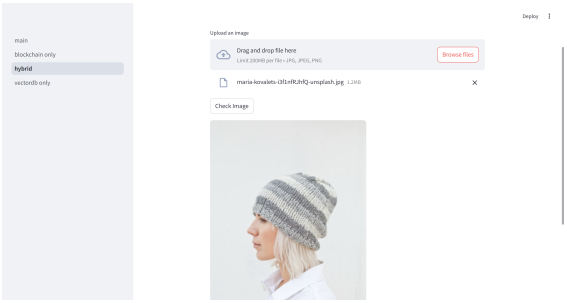
The EmbedAIDetect prototype has been designed to offer a simple and user-friendly interface. It follows a single-page application (SPA) architecture, where the interface dynamically updates content without requiring full page reloads. This results in a smoother user experience, faster interaction, and more seamless transitions between different functionalities within the system.

The prototype consists of uploading images, verifying via blockchain, scoring similarity using vector databases, and displaying the results in a unified interface. Users can easily navigate through different verification modes (blockchain-only, vector DB-only, hybrid) from the sidebar. Figure 4.4 shows screenshots of the prototype components.

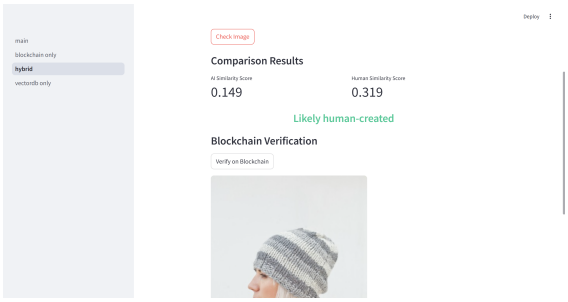
[1] `https://docs.streamlit.io/`



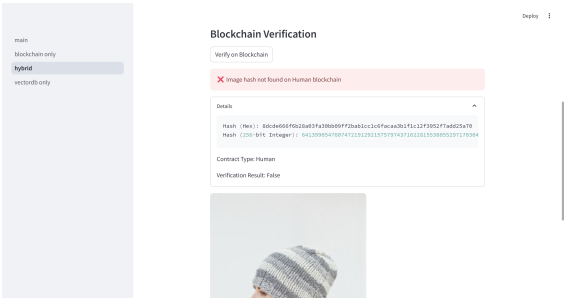
(a) Home page with system status and upload prompt



(b) Preview of the uploaded image



(c) Detection result with closest image match from database



(d) Blockchain verification result when image hash is not found

Figure 4.4: Screenshots of EmbedAIDetect User Interface

Chapter 5

Results and Evaluation

This chapter presents the results of the experimental design implemented to validate the thesis research and evaluate the EmbedAIDetect prototype, an AI-generated image detection system. The evaluation focuses on key performance metrics such as precision, recall, and precision, as well as estimated gas usage associated with the system's operations. In addition, feedback from research committee members is incorporated to assess the system's effectiveness in addressing the research questions.

5.1 Experimental Design

To rigorously assess the performance and robustness of EmbedAIDetect, we designed a comprehensive experimental framework focused on evaluating the role of vector embeddings in detecting AI-generated images. This framework addresses Research Question 2, which investigates the identification and implementation of an embedding technique that is generalizable and resilient to deliberate manipulations. The experimental setup consists of two key studies: one that explores vector similarity-based classification for detection and another that benchmarks various embedding models to determine their effectiveness under adversarial conditions.

All experiments were carried out in a high performance computing environment equipped with an Intel Core i9-14900K CPU, 66 GB of RAM, and an NVIDIA GeForce RTX 4090 GPU to ensure efficient handling of computationally intensive tasks. To evaluate the blockchain integration aspects of the system, we used Hardhat, a widely adopted Ethereum development framework, to simulate gas usage and smart contract interactions in a controlled environment.

5.1.1 Vector similarity based classification

The goal of this experiment is to evaluate the feasibility and effectiveness of using vector embeddings, coupled with cosine similarity, to distinguish between AI-generated and human-created images. The underlying hypothesis is that despite visual similarities, the embedding space representations of these images reveal distinct patterns when processed through pre-trained image classification models.

Dataset Construction

A diverse dataset was curated, excluding human faces to maintain generality. It comprises 9,000 AI-generated images and 6,074 human-created artworks. The AI images were generated using the Stable Diffusion 3.5 Medium model from Stability AI, available via the Hugging Face platform. To ensure uniformity and optimize both quality and generation time, all AI images were rendered at a fixed resolution of 512×512 pixels. The prompts were constructed systematically to cover a variety of themes, subjects, and artistic styles. The human data set is sourced from the Kaggle art-images collection, and data cleaning was performed to remove corrupted and duplicate entries.

Below are representative examples of prompts used to generate AI images:

- "A tropical floating islands in glitch art style"
- "A biotechnological deep space in photorealistic style"
- "A time travel rain forest canopy in psychedelic art style"
- "A neon concert hall in low poly style"
- "A mystical art studio in neon style"

The human-created images were sourced from the Kaggle art-images dataset and underwent careful preprocessing to remove duplicates and corrupted entries. Figure 5.1 illustrates sample images from both categories, highlighting the nature of the dataset and the types of manipulations applied.

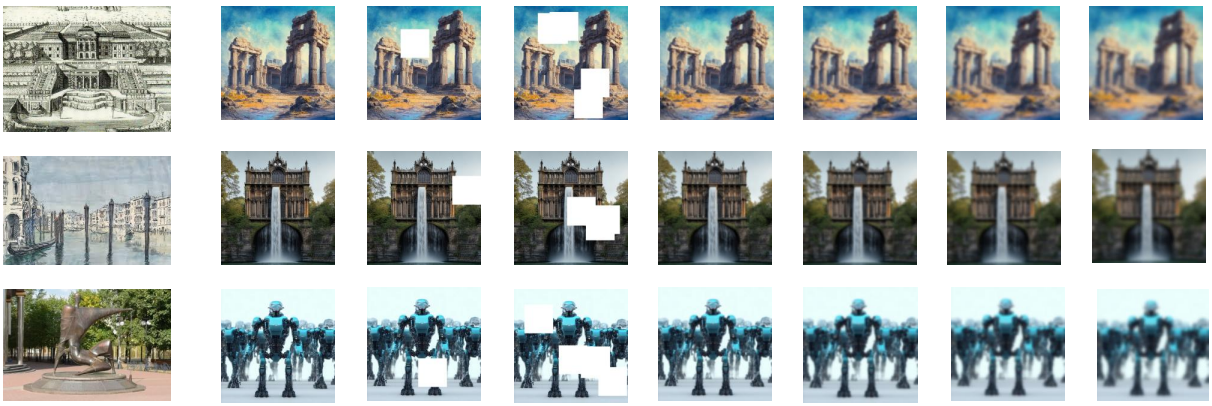


Figure 5.1: Sample images from human-art dataset and AI-generated image dataset with various modifications

Methodology

Our methodology implements a novel classification pipeline that involves extracting vector embeddings from each image using a selection of pre-trained models. These embeddings were stored in the Pinecone vector database, organized into distinct namespaces for training and testing datasets, which were split in a 4:1 ratio. For each image in the test set, the closest distance to the AI-generated image training set and the human-created image training set is calculated using a similarity search with the cosine distance as the metric. These nearest distances are stored in spreadsheets for further evaluation.

Table 5.1: Details of pre-trained models used for embedding extraction

Model Name	**Publisher**	**Model Identifier**	**Embedding Dimension**
CLIP	OpenAI	`openai/clip-vit-base-patch32`	512
ViT	Google	`google/vit-base-patch16-224`	768
DINOv2	Facebook	`facebook/dinov2-base`	768
ResNet-50	Microsoft	`microsoft/resnet-50`	2048
AIMv2	Apple	`apple/aim-v2-base`	1024

The experiments were carried out repeatedly using five pre-trained models, as shown in Table 5.1. For each model, separate Pinecone indexes were maintained, and the data was logically partitioned to isolate training and test sets. After computing the cosine distances between test image embeddings and both AI-generated and human-created training sets, a simple decision rule was applied: a test image was classified as AI-generated if its closest match belonged to the AI set and as human-created otherwise.

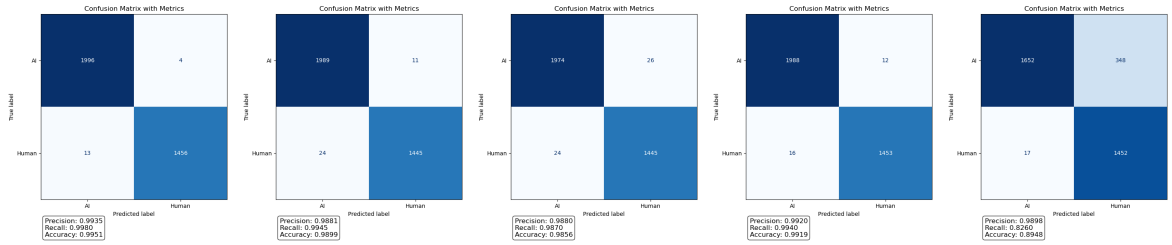
This classification logic represents the central hypothesis of the detection system. The resulting distance values were recorded in two spreadsheets—one for AI test images and one for human test images, and used to generate the confusion matrix and related evaluation metrics such as precision, recall and accuracy.

The classification rule is defined formally as follows.

$$\text{AI\_or\_not}(x) = \begin{cases} 1 & \text{if } d(x, \mathcal{A}) \leq d(x, \mathcal{H}) \\ 0 & \text{if } d(x, \mathcal{A}) > d(x, \mathcal{H}) \end{cases} \tag{5.1}$$

where;

$$\begin{aligned} x &: \text{supplied image embedding vector} \\ \mathcal{A} &: \text{set of AI-generated image embeddings} \\ \mathcal{H} &: \text{set of human-generated image embeddings} \\ d(x, S) &= \min_{s \in S} \{\text{cosine\_distance}(x, s)\} \end{aligned}$$



(a) CLIP Confusion Matrix (b) DINOV2 Confusion Matrix (c) Google VIT Confusion Matrix (d) AIMV2 Confusion Matrix (e) ResNet50 Confusion Matrix

Figure 5.2: Confusion matrices for the vector similarity-based classification experiment

Results

Figure 5.2 presents confusion matrices for five different models (CLIP, DINOv2, Google ViT, AIMV2, and ResNet50), which illustrate their performance in classifying images as AI-generated or human-created. Confusion matrices show the true positives, true negatives, false positives, and false negatives, providing insight into how well the models distinguish between the two categories. A higher number of true positives and true negatives, with fewer false positives and negatives, indicates better model performance.

The results, summarized in Table 5.2, show consistently high performance in most models. OpenAI's CLIP achieved the highest accuracy at 99.51%, with precision and recall nearing perfect scores. AIMv2 also demonstrated robust performance across all metrics. Although the ResNet-50 model maintained high precision, its lower recall (82.6%) resulted in an overall accuracy of 89.48%, highlighting its sensitivity to false negatives.

Table 5.2: Performance metric results across five selected embedding models

Models	**Precision**	**Recall**	**Accuracy**
CLIP	0.9935	0.998	0.9951
ViT	0.988	0.987	0.9856
DINOv2	0.9881	0.9945	0.9899
ResNet-50	0.9898	0.826	0.8948
AIMv2	0.992	0.994	0.9919

5.1.2 Benchmarking different embedding models

The goal of this experiment is to evaluate the robustness and consistency of various image embedding models when subjected to deliberate manipulations of input images. The primary objective is to determine whether the embeddings of altered images remain sufficiently similar to their original counterparts, thus validating the generalizability of embedding techniques used for the EmbedAIDetect system. This analysis provides critical evidence to support the

system's ability to withstand adversarial or heavy image modifications, as stated in Research Question 2.

Experimental Setup

Using the original dataset of 9,000 AI-generated images from the previous experiment, we created six distinct sets of modified images to simulate common geometric and visual distortions. These included single- and multiple-white patch overlays, as well as resolution reduction (downsampling). Each of the six modifications generated a corresponding set of 9,000 altered images.

For each modification, embeddings were computed using the five pre-trained models introduced earlier: OpenAI CLIP, Google ViT, Facebook DINOv2, Microsoft ResNet-50, and Apple AIMv2. Embeddings of the modified images were stored in separate Pinecone indexes and compared against the embeddings of the original, unaltered images using cosine similarity. A correct match is counted when the nearest neighbor in the embedding space corresponds to the unmodified version of the image. The accuracy is calculated using the following equation.

$$\text{Accuracy} = \frac{\text{Number of Correct Matches}}{9000} \times 100\% \tag{5.2}$$

where a correct match occurs when the modified image's
embedding is closest to its original AI image embedding

This approach directly measures whether the model preserves semantic similarity between the original and manipulated images in the embedding space.

Results: Geometric modifications

Table 5.3 summarizes the accuracy of each model under three types of geometric alterations: a single white patch (128×128 pixels), multiple patches (3–5), and reduction in resolution (from 512×512 to 128×128 pixels). Facebook's DINOv2 exhibited the highest resilience, achieving perfect accuracy with single patch overlays and 99.94% accuracy under resolution reduction. In contrast, OpenAI's CLIP showed the greatest performance drop under multiple patch overlays, with accuracy falling to 73.34%.

Table 5.3: Model accuracy (%) for patch overlays and downsampling

Models	**1 Patch Overlay**	**3-5 Patches Overlay**	**Resizing**
CLIP	97.26	73.34	98.04
Google VIT	99.88	91.62	98.98
DinoV2	100.00	99.62	99.94
ResNet-50	99.89	84.02	98.47
AIMV2	99.96	92.59	99.81

Results: Blur transformations

The second set of modifications applied varying levels of Gaussian blur to the original images, ranging from intensity 20% to intensity 80%. The objective was to observe degradation patterns in the accuracy of the matching as the visual noise increased.

Table 5.4: Model accuracy (%) under various blur intensities

Models	**Blur Intensity**			
	20%	**40%**	**60%**	**80%**
CLIP	96.68	80.96	48.12	30.02
Google ViT	99.86	96.00	84.17	67.06
DinoV2	100.00	99.87	96.86	88.44
ResNet-50	98.52	74.18	44.36	24.67
AIMv2	99.65	96.81	90.08	80.13

As detailed in Table 5.4, DINOv2 once again demonstrated superior robustness, maintaining 88.44% accuracy even at the highest blur level. Transformer-based models such as AIMv2 and Google ViT also performed well at lower blur levels, exceeding 99% accuracy at 20% intensity. However, all models exhibited a gradual decline in accuracy with increased blur. In particular, ResNet-50 showed the steepest performance degradation, dropping to just 24.67% accuracy at 80% blur.

5.1.3 Smart Contract Gas Usage Estimation

This experiment was designed to evaluate and compare the gas consumption incurred when storing 256-bit embedding hashes on the Ethereum blockchain using two different Solidity data types: `uint256` and `string`. The purpose was to empirically determine the gas efficiency of each approach, which is critical when scaling decentralized applications that store or verify large volumes of hashed data.

Experimental Setup

To empirically assess gas usage differences, two separate Solidity smart contracts were implemented:

- HashStorageInt: Stores the 256-bit image embedding hash as uint256.
- HashStorageStr: Stores the same 256-bit hash as a string.

Each smart contract included two core functions: `storeHash(input_hash)` for storing a hash value on the blockchain, and `hashExists(input_hash)` for verifying whether a given hash is already present in the contract's storage.

Both contracts were developed and tested in a local Hardhat Ethereum environment, where extensive unit tests were written to validate the functionality of the smart contracts.

Subsequently, both contracts were also deployed on the Sepolia testnet, where transactions were executed to measure real-world gas usage. Only 200 image embedding hashes were stored on Sepolia due to constraints in transaction time and limited gas availability.

Results

Gas usage was recorded for each transaction involving the `storeHash(input_hash)` function in both contracts. The summarized statistics are provided in Table 5.5. These findings clearly show that storing a 256-bit embedding hash as a uint256 is approximately 2.7 times more gas efficient than storing it as a string.

Table 5.5: Gas usage statistics for hash storage using `uint256` and `string` types

Statistic	**Gas Usage (`uint256`)**	**Gas Usage (`string`)**
Count	9,000 transactions	9,000 transactions
Mean	36,207 gas	97,667 gas
Median	36,184 gas	97,667 gas
Minimum	21,528 gas	97,667 gas
Maximum	51,228 gas	97,667 gas

5.2 Prototype Evaluation

To assess the practical performance of our AI image detection system, we developed and deployed a working prototype capable of performing image classification based on stored image embeddings and integration of smart contracts. The goal of this evaluation is to understand how the system behaves when given input images from the real world and whether it can accurately categorize images as AI-generated or human-generated. The prototype allows for uploading any image, processes it through an embedding model, and returns a classification result based on the similarity match with the embeddings stored in the local database.

This evaluation phase focused not only on the overall classification accuracy, but also on identifying potential limitations, especially in diverse and uncontrolled image settings. The prototype was subjected to various image categories to simulate practical use cases. We specifically examined how the system handles facial images (often generated by StyleGAN) compared to non-facial or generic image categories, such as artwork and nature.

5.2.1 Prototype 1: Blockchain only Evaluation

To evaluate the blockchain-only prototype, we conducted manual testing using the system's user interface by uploading individual image files for classification. In this setup, the backend system used a predefined dataset of 7,000 AI-generated images and 7,000 real (human) images. For each image, an embedding was generated using the DINOv2 model, and a

256-bit cryptographic hash of this embedding was computed and stored on the Ethereum Sepolia test network via two dedicated smart contracts: `HashStorageAI` for AI-generated images and `HashStorageHuman` for real ones. When a new image is uploaded, the system computes its embedding and hash, then compares the hash against both smart contracts. If a match is found in either contract, the image is classified accordingly—AI or human—and marked as verifiable. Otherwise, if no match exists, the system cannot make a definitive classification, and the result is labeled as inconclusive.

This evaluation method, although functionally sound in design, encountered several practical challenges due to limitations inherent in on-chain data storage. Specifically, storing all 14,000 image hashes on the Ethereum blockchain introduced high computational and financial overhead. Every transaction to add a hash to a smart contract incurs a gas fee, and when many transactions are sent in rapid succession, network congestion and rate limitations often lead to failed or dropped transactions. From a technical standpoint, this results in nonce conflicts, out-of-gas errors, or rate-limiting by the node provider (Alchemy), thereby preventing the successful storage of all intended hashes. Consequently, while every image embedding was stored in the local vector database, only a subset of those corresponding hashes could be reliably registered on-chain. As a result, many uploaded images during evaluation matched an embedding in the database but failed the blockchain verification check—leading to false negatives in verifiability. This limitation highlights the trade-offs between blockchain transparency and system scalability, particularly in use cases involving large-scale data operations.

5.2.2 Prototype 2: Database only Evaluation

The second prototype focused solely on embedding-based classification using vector similarity search. Before formally evaluating Prototype 2, we expanded the system's internal database to better support classification of facial images—an area where the system initially struggled. We first introduced 10,000 high-resolution AI-generated facial images created using StyleGAN, sourced from `https://thispersondoesnotexist.com/`, into the AI category. The model showed strong performance in identifying similar StyleGAN images during testing, correctly labeling over 90% of such inputs as synthetic. Some examples of correctly classified AI faces are shown in Figure 5.3. However, this also led to a growing tendency to misclassify real human faces as AI-generated, suggesting the embedding space was becoming biased toward synthetic features.

To mitigate this issue, we integrated 1,441 real facial images from the Chicago Face Database (CFD) [108, 109, 110] into the human category. This addition provided more diversity in real facial representations and slightly improved accuracy for some real images during early tests. However, the underlying issue persisted: the model continued to misclassify many real human faces, particularly those with uniform lighting and frontal poses, likely because of their visual similarity to StyleGAN outputs. This imbalance in the database influenced how the system later performed during the formal evaluation stage.

With the internal database prepared, we proceeded to evaluate the prototype using two external datasets: (1) CIFAKE [99, 111], which includes a wide range of non-facial real and

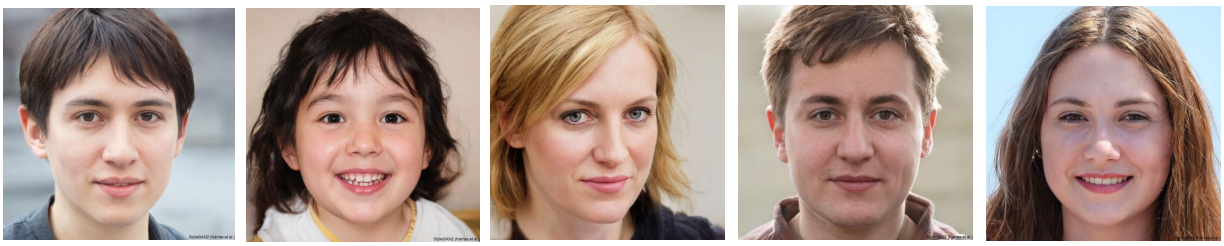


Figure 5.3: Sample StyleGAN-generated synthetic images used for EmbedAIDetect prototype evaluation

AI-generated images, and (2) 140k Real and Fake Faces [112], which contains high-resolution facial images. The goal was to evaluate how accurately the system could classify input images as AI-generated or real based on similarity to pre-stored embeddings using cosine distance. The system was tested using Python scripts that emulated the classification flow of the application, including embedding computation, vector similarity querying, and prediction logging. Results were saved in CSV format with fields such as filename, true label, similarity scores, and predicted label. A snippet of this output is shown below:

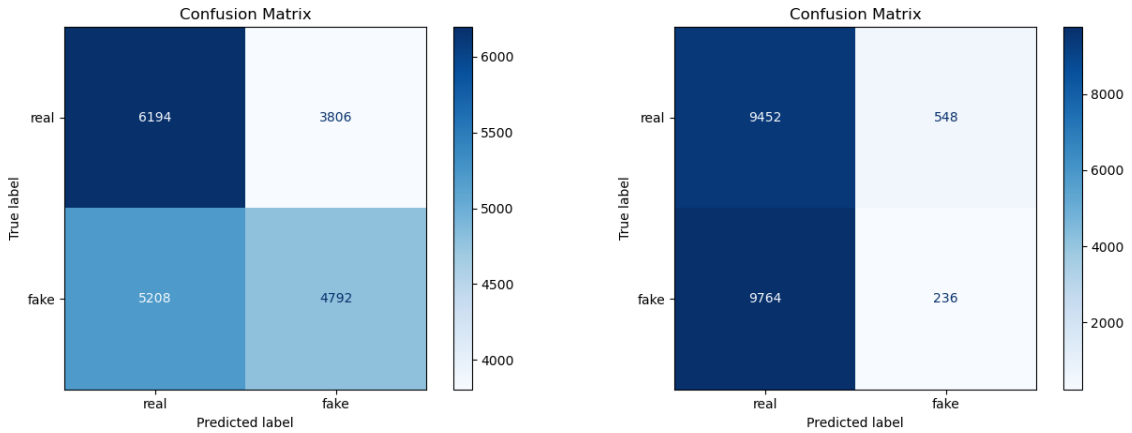
```
filename,true_label,human_similarity,ai_similarity,predicted_label
0375 (7).jpg,real,0.2266,0.1677,real
0838 (7).jpg,real,0.2846,0.2476,real
0174 (5).jpg,real,0.4529,0.5668,fake
0445 (3).jpg,real,0.1606,0.1622,fake
0771 (9).jpg,real,0.2502,0.3689,fake
```

We then computed standard classification metrics such as accuracy, precision, recall, and F1-score, along with confusion matrices, to assess how the system performed across both facial and non-facial image categories.

Quantitative Results and Observations

On the CIFAKE dataset, which contains objects, scenes, and non-facial artwork, the system showed balanced performance. As shown in Figure 5.4a, it correctly classified 6,194 out of 10,000 real images and 4,792 out of 10,000 fake images. While not highly accurate, the results indicate that the model can reasonably distinguish between AI-generated and real content in general image categories where visual patterns are more distinct and less uniform.

In contrast, the evaluation on the 140k Real and Fake Faces dataset highlighted significant limitations. As shown in Figure 5.4b, the system correctly labeled most real facial images (9,452 out of 10,000), but misclassified nearly all fake faces: only 236 were correctly identified as AI-generated, while 9,764 were incorrectly predicted as real. This imbalance reflects a strong bias toward labeling facial content as real, likely due to the embedding model's



(a) Confusion matrix for CIFAKE dataset evaluated using Prototype 2

(b) Confusion matrix for 140k Real and Fake Faces dataset evaluated using Prototype 2

Figure 5.4: Confusion matrices showing classification performance of Prototype 2 using vector similarity search

inability to capture subtle generative artifacts present in StyleGAN outputs, especially when those features overlap with realistic cues found in the CFD dataset.

These results reinforce a key insight: While embedding-based similarity is reasonably effective for broad semantic differences (as seen in non-facial images), it falls short when used to detect highly realistic facial forgeries. This gap directly influenced the decision to incorporate blockchain-based verification in the hybrid prototype, with the aim of enhancing confidence and trust in high-risk classifications.

5.2.3 Prototype 3: Hybrid Approach Evaluation

Prototype 3 combines the embedding-based similarity classification from Prototype 2 with the blockchain verification mechanism from Prototype 1. As it does not introduce new classification logic, but rather integrates existing components sequentially, first identifying the closest match via vector similarity, then checking its hash on-chain, no separate evaluation was conducted. Its behavior and performance characteristics are directly inherited from the earlier prototypes. The primary goal of this version is to enhance trust and verifiability in the results, especially in edge cases, by confirming that the matched embedding has an immutable record on the blockchain. While not formally benchmarked, the hybrid workflow was manually validated through the user interface and confirmed to operate as expected.

Chapter 6

Conclusion

The thesis research presents a novel concept of identifying and detecting AI-generated images. The findings suggest that despite extensive manipulations, the semantic meaning of images remains similar to their original counterparts, allowing for identification. However, the efficacy of this approach is inherently limited by the dataset used for embedding generation. Also, the widespread adoption of this system across big technology companies such as OpenAI, Meta, and Google is crucial to ensuring its success. Standardization and a shared database of vector embeddings are necessary because different embedding models produce varying representations. Future work will focus on using IPFS to establish decentralized image storage. By ensuring the integrity and accessibility of vector embeddings, such a system could pave the way for a more robust and universally accepted AI image detection framework. Ultimately, collaboration across the industry will be vital in creating a scalable, transparent, and effective solution for combating the challenges posed by AI-generated media.

6.1 Limitations

This thesis, while contributing valuable insights, has several limitations.

- The model's performance was constrained by the size and diversity of the training dataset, which may not fully represent the range of evolving tampering techniques, affecting generalization to novel manipulations.
- Computational limitations restricted exploration of more complex deep learning models, potentially capping detection performance.
- Evaluation under controlled conditions may not reflect real-world environments, where compression, manual tampering, and platform distortions can affect accuracy.
- Reliance on visual features alone, without multimodal analysis like metadata or forensic traces, limits detection of sophisticated forgeries such as GAN-generated images and deepfakes.
- Finally, biases in the design of the data set, the selection of algorithms and the evaluation may have subtly influenced the results despite validation efforts.

6.2 Future Work

Some of the possible future research work and improvements are discussed below:

- Future work involves making the model and system more robust by incorporating more varied datasets and focusing on training and testing models directly instead of relying solely on pre-trained models.
- Another direction is to add explainability and interpretability by displaying the system's source data, clearly illustrating why an image was flagged as AI-generated or human-created.
- A scalable solution for image provenance is required, as the storage limitations and increasing costs associated with smart contract-based hash verification present challenges as the number of stored hashes grows.
- Future improvements should consider leveraging PRNU analysis and associated metadata like EXIF data, to further enhance classification accuracy.

Appendix A

Smart Contracts

This chapter includes the smart contracts used in the EmbedDetect system.

A.1 Smart Contract for AI Image Embedding Hash

The following smart contract is responsible for storing and verifying AI-generated image hashes using a dynamic array on the blockchain.

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

contract HashStorageAI {
    // Dynamic array to store hashes as uint256
    uint256[] private ai_hash_list;

    // Event to emit when a new hash is stored
    event HashStored(uint256 hash, uint256 hash_list_len);

    // Function to store a single hash
    function storeHash(uint256 _hash) public {
        ai_hash_list.push(_hash);
        emit HashStored(_hash, ai_hash_list.length);
    }

    // Function to get total number of hashes stored
    function getTotalHashes() public view returns (uint256) {
        return ai_hash_list.length;
    }

    function hashExists(uint256 _hash) public view returns (bool) {
        for (uint256 i = 0; i < ai_hash_list.length; i++) {
            if (ai_hash_list[i] == _hash) {
                return true;
            }
        }
        return false;
```

```
    }
}
```

A.2 Smart Contract for Human-created Image Embedding Hash

The following contract is responsible for storing and verifying hashes associated with human-created images.

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

contract HashStorageHuman {
    // Dynamic array to store hashes as uint256
    uint256[] private human_hash_list;

    // Event to emit when a new hash is stored
    event HashStored(uint256 hash, uint256 hash_list_len);

    // Function to store a single hash
    function storeHash(uint256 _hash) public {
        human_hash_list.push(_hash);
        emit HashStored(_hash, human_hash_list.length);
    }

    // Function to get total number of hashes stored
    function getTotalHashes() public view returns (uint256) {
        return human_hash_list.length;
    }

    function hashExists(uint256 _hash) public view returns (bool) {
        for (uint256 i = 0; i < human_hash_list.length; i++) {
            if (human_hash_list[i] == _hash) {
                return true;
            }
        }
        return false;
    }
}
```

References

[1] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. *Advances in neural information processing systems*, 33:6840–6851, 2020.

[2] Alexander Quinn Nichol and Prafulla Dhariwal. Improved denoising diffusion probabilistic models. In *International conference on machine learning*, pages 8162–8171. PMLR, 2021.

[3] Staphord Bengesi, Hoda El-Sayed, Md Kamruzzaman Sarker, Yao Houkpati, John Irungu, and Timothy Oladunni. Advancements in generative ai: A comprehensive review of gans, gpt, autoencoders, diffusion model, and transformers. *IEEE Access*, 2024.

[4] Aditya Ramesh, Mikhail Pavlov, Gabriel Goh, Scott Gray, Chelsea Voss, Alec Radford, Mark Chen, and Ilya Sutskever. Zero-shot text-to-image generation. In *International conference on machine learning*, pages 8821–8831. Pmlr, 2021.

[5] Zach Winn. Pushing the frontiers of art and technology with generative ai. `https://news.mit.edu/2023/pushing-frontiers-art-technology-generative-ai-1129`, 2023. Accessed: June 9, 2024.

[6] S. Mitra Kalita and Anand Iyer. How generative ai could disrupt creative work. `https://hbr.org/2023/04/how-generative-ai-could-disrupt-creative-work`, 2023. Accessed: June 9, 2024.

[7] Abhijeeth Madhu. Survey reveals 9 out of 10 artists believe current copyright laws are outdated in the age of generative ai technology. `https://bookanartist.co/blog/2023-artists-survey-on-ai-technology/`, 2023. Accessed: June 9, 2024.

[8] Kevin Roose. An a.i.-generated picture won an art prize. artists aren't happy. `https://www.nytimes.com/2022/09/02/technology/ai-artificial-intelligence-artists.html`, 2022. Accessed: June 9, 2024.

[9] PBS NewsHour. Fake ai images of putin, trump being arrested spread online. `https://www.pbs.org/newshour/politics/fake-ai-images-of-putin-trump-being-arrested-spread-online`, 2023. Accessed: June 9, 2024.

[10] Neha Sandotra and Bhavna Arora. A comprehensive evaluation of feature-based ai techniques for deepfake detection. *Neural Computing and Applications*, 36(8):3859–3887, 2024.

[11] Zhengyuan Jiang, Jinghuai Zhang, and Neil Zhenqiang Gong. Evading watermark based detection of ai-generated content. In *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*, pages 1168–1181, 2023.

[12] Frank Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.

[13] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533–536, 1986.

[14] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436–444, 2015.

[15] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[16] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25, 2012.

[17] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[18] 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.

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

[20] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[21] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[22] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[23] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.

[24] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[25] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

[26] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. *Advances in neural information processing systems*, 27, 2014.

[27] Dor Bank, Noam Koenigstein, and Raja Giryes. Autoencoders. *Machine learning for data science handbook: data mining and knowledge discovery handbook*, pages 353–374, 2023.

[28] Geoffrey E Hinton and Ruslan R Salakhutdinov. Reducing the dimensionality of data with neural networks. *science*, 313(5786):504–507, 2006.

[29] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[30] Solomon Kullback and Richard A Leibler. On information and sufficiency. *The annals of mathematical statistics*, 22(1):79–86, 1951.

[31] George Lawton. What is generative ai? everything you need to know. `https://www.techtarget.com/searchenterpriseai/definition/generative-AI`, 2023. Accessed: June 9, 2024.

[32] Kim Martineau. What is generative ai? `https://research.ibm.com/blog/what-is-generative-AI`, 2023. Accessed: June 9, 2024.

[33] OpenAI. Dall·e: Creating images from text. `https://openai.com/index/dall-e/`, 2021. Accessed: June 9, 2024.

[34] OpenAI. Chatgpt. `https://openai.com/index/chatgpt/`, 2023. Accessed: June 9, 2024.

[35] Pandu Nayak. Mum: A new ai milestone for understanding information. `https://blog.google/products/search/introducing-mum/`, 2021. Accessed: June 9, 2024.

[36] OpenAI. Gpt-4. https://openai.com/index/gpt-4-research/, 2023. Accessed: June 9, 2024.

[37] Demis Hassabis and Sundar Pichai. Introducing gemini: Google's most capable ai model yet. https://blog.google/technology/ai/google-gemini-ai/, 2023. Accessed: June 9, 2024.

[38] Yusuf Mehdi. Reinventing search with a new ai-powered microsoft bing and edge, your copilot for the web. https://blogs.microsoft.com/blog/2023/02/07/reinventing-search-with-a-new-ai-powered-microsoft-bing-and-edge-your-copilot-for- 2023. Accessed: June 9, 2024.

[39] Youssef Hosni. Comprehensive introduction to ai image generation, 2023. Accessed on June 5, 2024.

[40] AltexSoft. Generative ai: How ai image generators work. https://www.altexsoft.com/blog/ai-image-generation/, 2023. Accessed: June 2024.

[41] Aditya Ramesh, Mikhail Pavlov, Gabriel Goh, Scott Gray, and OpenAI. Dall·e: Creating images from text, 2021. Accessed on June 5, 2024.

[42] Alex McFarland. Beginner's guide to ai image generators, 2023. Accessed on June 5, 2024.

[43] Chester Avey. Ethics of ai image generation, December 2023. Accessed on June 5, 2024.

[44] AI or Not. How can ai-generation photos harm each of us, 2023. Accessed on June 5, 2024.

[45] 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.

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

[47] Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A Efros. Image-to-image translation with conditional adversarial networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1125–1134, 2017.

[48] Christian Ledig, Lucas Theis, Ferenc Huszár, Jose Caballero, Andrew Cunningham, Alejandro Acosta, Andrew Aitken, Alykhan Tejani, Johannes Totz, Zehan Wang, et al. Photo-realistic single image super-resolution using a generative adversarial network. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4681–4690, 2017.

[49] Aäron Van Den Oord, Nal Kalchbrenner, and Koray Kavukcuoglu. Pixel recurrent neural networks. In *International conference on machine learning*, pages 1747–1756. PMLR, 2016.

[50] Xi Chen, Nikhil Mishra, Mostafa Rohaninejad, and Pieter Abbeel. Pixelsnail: An improved autoregressive generative model. In *International conference on machine learning*, pages 864–872. PMLR, 2018.

[51] Jacob Menick and Nal Kalchbrenner. Generating high fidelity images with subscale pixel networks and multidimensional upscaling. *arXiv preprint arXiv:1812.01608*, 2018.

[52] Leon A Gatys, Alexander S Ecker, and Matthias Bethge. A neural algorithm of artistic style. *arXiv preprint arXiv:1508.06576*, 2015.

[53] Aston Zhang, Zack C. Lipton, Mu Li, and Alex J. Smola. Neural style transfer, 2020. Accessed on June 9, 2024.

[54] Justin Johnson, Alexandre Alahi, and Li Fei-Fei. Perceptual losses for real-time style transfer and super-resolution. In *Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11-14, 2016, Proceedings, Part II 14*, pages 694–711. Springer, 2016.

[55] Xun Huang and Serge Belongie. Arbitrary style transfer in real-time with adaptive instance normalization. In *Proceedings of the IEEE international conference on computer vision*, pages 1501–1510, 2017.

[56] Jascha Sohl-Dickstein, Eric Weiss, Niru Maheswaranathan, and Surya Ganguli. Deep unsupervised learning using nonequilibrium thermodynamics. In *International conference on machine learning*, pages 2256–2265. PMLR, 2015.

[57] Yang Song and Stefano Ermon. Generative modeling by estimating gradients of the data distribution. *Advances in neural information processing systems*, 32, 2019.

[58] Hugging Face. Diffuse the rest. `https://huggingface.co/spaces/huggingface-projects/diffuse-the-rest`, 2023. Accessed: June 9, 2024.

[59] Emad Mostaque. Tweet by emad mostaque. `https://x.com/EMostaque/status/1587844074064822274?lang=en&mx=2`, 2022. Accessed: June 9, 2024.

[60] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. Stable diffusion. `https://github.com/CompVis/stable-diffusion`, 2022. Accessed: June 9, 2024.

[61] 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*, pages 10684–10695, 2022.

[62] Patrick Esser, Sumith Kulal, Andreas Blattmann, Rahim Entezari, Jonas Müller, Harry Saini, Yam Levi, Dominik Lorenz, Axel Sauer, Frederic Boesel, et al. Scaling rectified flow transformers for high-resolution image synthesis. In *Forty-first International Conference on Machine Learning*, 2024.

[63] Wikipedia. Stable diffusion. `https://en.wikipedia.org/wiki/Stable_Diffusion`, 2024. Accessed: June 9, 2024.

[64] Tomas Mikolov, Kai Chen, Gregory S. Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. In *International Conference on Learning Representations*, 2013.

[65] 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.

[66] Zhou Wang, Alan C Bovik, Hamid R Sheikh, and Eero P Simoncelli. Image quality assessment: from error visibility to structural similarity. *IEEE transactions on image processing*, 13(4):600–612, 2004.

[67] Diego Gragnaniello, Francesco Marra, and Luisa Verdoliva. Detection of ai-generated synthetic faces. In *Handbook of digital face manipulation and detection: From deepfakes to morphing attacks*, pages 191–212. Springer International Publishing Cham, 2022.

[68] Zhiyuan He, Pin-Yu Chen, and Tsung-Yi Ho. Rigid: A training-free and model-agnostic framework for robust ai-generated image detection. *arXiv preprint arXiv:2405.20112*, 2024.

[69] Zihan Liu, Hanyi Wang, Yaoyu Kang, and Shilin Wang. Mixture of low-rank experts for transferable ai-generated image detection. *arXiv preprint arXiv:2404.04883*, 2024.

[70] Amit Singhal et al. Modern information retrieval: A brief overview. *IEEE Data Eng. Bull.*, 24(4):35–43, 2001.

[71] Jianguo Wang, Xiaomeng Yi, Rentong Guo, Hai Jin, Peng Xu, Shengjun Li, Xiangyu Wang, Xiangzhou Guo, Chengming Li, Xiaohai Xu, Kun Yu, Yuxing Yuan, Yinghao Zou, Jiquan Long, Yudong Cai, Zhenxiang Li, Zhifeng Zhang, Yihua Mo, Jun Gu, Ruiyi Jiang, Yi Wei, and Charles Xie. Milvus: A purpose-built vector data management system. In *Proceedings of the 2021 International Conference on Management of Data*, SIGMOD '21, page 2614–2627, New York, NY, USA, 2021. Association for Computing Machinery.

[72] Artem Babenko Yandex and Victor Lempitsky. Efficient indexing of billion-scale datasets of deep descriptors. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2055–2063, 2016.

[73] Qi Chen, Haidong Wang, Mingqin Li, Gang Ren, Scarlett Li, Jeffery Zhu, Jason Li, Chuanjie Liu, Lintao Zhang, and Jingdong Wang. Sptag: A library for fast approximate nearest neighbor search, 2018.

[74] Christian Garcia-Arellano, Hamdi Roumani, Richard Sidle, Josh Tiefenbach, Kostas Rakopoulos, Imran Sayyid, Adam Storm, Ronald Barber, Fatma Ozcan, Daniel Zilio, et al. Db2 event store: a purpose-built iot database engine. *Proceedings of the VLDB Endowment*, 13(12):3299–3312, 2020.

[75] Jeff Johnson, Matthijs Douze, and Hervé Jégou. Billion-scale similarity search with gpus. *IEEE Transactions on Big Data*, 7(3):535–547, 2019.

[76] Satoshi Nakamoto and A Bitcoin. A peer-to-peer electronic cash system. *Bitcoin.–URL: https://bitcoin. org/bitcoin. pdf*, 4(2):15, 2008.

[77] Ethereum Foundation. Introduction to ethereum. `https://ethereum.org/en/developers/docs/intro-to-ethereum/`, 2024. Accessed: April 4, 2025.

[78] Nick Szabo. Smart contracts. `https://www.fon.hum.uva.nl/rob/Courses/InformationInSpeech/CDROM/Literature/LOTwinterschool2006/szabo.best.vwh.net/smart.contracts.html`, 1994. Accessed April 20, 2025.

[79] Solidity Team. Solidity v0.8.25 documentation. `https://docs.soliditylang.org/en/v0.8.25/`, 2024. Released March 14, 2024. Accessed April 20, 2025.

[80] Ethereum Foundation. Gas and fees: Technical overview. `https://ethereum.org/en/developers/docs/gas/`, 2025. Last updated February 25, 2025. Accessed April 20, 2025.

[81] Wikipedia contributors. Cryptocurrency wallet. `https://en.wikipedia.org/wiki/Cryptocurrency_wallet`, 2025. Accessed April 20, 2025.

[82] Consensys Software Inc. Metamask: The leading crypto wallet platform. `https://metamask.io/`, 2025. Accessed April 20, 2025.

[83] Suvarna Sharma, Puneeta Rosmin, and Amit Bhagat. Blockchain technology: Limitations and future possibilities. In *Blockchain Applications in IoT Security*, pages 140–151. IGI Global, 2021.

[84] Pablo Pernias, Dominic Rampas, Mats L. Richter, Christopher J. Pal, and Marc Aubreville. Wuerstchen: An efficient architecture for large-scale text-to-image diffusion models, 2023.

[85] Jinhyeok Jang, Chan-Hyun Youn, Minsu Jeon, and Changha Lee. Rethinking peculiar images by diffusion models: Revealing local minima's role. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 2454–2461, 2024.

[86] Allan Kouidri. Top ai diffusion models: Ultimate comparison & guide [2024], 2024. Accessed on June 9, 2024.

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

[88] Jonathan Ho, Chitwan Saharia, William Chan, David J Fleet, Mohammad Norouzi, and Tim Salimans. Cascaded diffusion models for high fidelity image generation. *Journal of Machine Learning Research*, 23(47):1–33, 2022.

[89] Lvmin Zhang, Anyi Rao, and Maneesh Agrawala. Adding conditional control to text-to-image diffusion models. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 3836–3847, 2023.

[90] William Peebles and Saining Xie. Scalable diffusion models with transformers. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 4195–4205, 2023.

[91] Riccardo Corvi, Davide Cozzolino, Giada Zingarini, Giovanni Poggi, Koki Nagano, and Luisa Verdoliva. On the detection of synthetic images generated by diffusion models. In *ICASSP 2023-2023 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 1–5. IEEE, 2023.

[92] Utkarsh Ojha, Yuheng Li, and Yong Jae Lee. Towards universal fake image detectors that generalize across generative models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 24480–24489, 2023.

[93] Muhammad Asad Arshed, Ayed Alwadain, Rao Faizan Ali, Shahzad Mumtaz, Muhammad Ibrahim, and Amgad Muneer. Unmasking deception: Empowering deepfake detection with vision transformer network. *Mathematics*, 11(17):3710, 2023.

[94] Yan Ju, Shan Jia, Jialing Cai, Haiying Guan, and Siwei Lyu. Glff: Global and local feature fusion for ai-synthesized image detection. *IEEE Transactions on Multimedia*, 2023.

[95] Samah S Baraheem and Tam V Nguyen. Ai vs. ai: Can ai detect ai-generated images? *Journal of Imaging*, 9(10):199, 2023.

[96] Fernando Martin-Rodriguez, Rocio Garcia-Mojon, and Monica Fernandez-Barciela. Detection of ai-created images using pixel-wise feature extraction and convolutional neural networks. *Sensors*, 23(22):9037, 2023.

[97] Ziyi Xi, Wenmin Huang, Kangkang Wei, Weiqi Luo, and Peijia Zheng. Ai-generated image detection using a cross-attention enhanced dual-stream network. In *2023 Asia Pacific Signal and Information Processing Association Annual Summit and Conference (APSIPA ASC)*, pages 1463–1470. IEEE, 2023.

[98] Pierre Fernandez, Guillaume Couairon, Hervé Jégou, Matthijs Douze, and Teddy Furon. The stable signature: Rooting watermarks in latent diffusion models. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 22466–22477, 2023.

[99] Jordan J Bird and Ahmad Lotfi. Cifake: Image classification and explainable identification of ai-generated synthetic images. *IEEE Access*, 2024.

[100] Muhammad Asad Arshed, Shahzad Mumtaz, Muhammad Ibrahim, Christine Dewi, Muhammad Tanveer, and Saeed Ahmed. Multiclass ai-generated deepfake face detection using patch-wise deep learning model. *Computers*, 13(1):31, 2024.

[101] Jiaxuan Chen, Jieteng Yao, and Li Niu. A single simple patch is all you need for ai-generated image detection. *arXiv preprint arXiv:2402.01123*, 2024.

[102] Mingjian Zhu, Hanting Chen, Qiangyu Yan, Xudong Huang, Guanyu Lin, Wei Li, Zhijun Tu, Hailin Hu, Jie Hu, and Yunhe Wang. Genimage: A million-scale benchmark for detecting ai-generated image. *Advances in Neural Information Processing Systems*, 36, 2024.

[103] Yanhao Li, Quentin Bammey, Marina Gardella, Tina Nikoukhah, Jean-Michel Morel, Miguel Colom, and Rafael Grompone Von Gioi. Masksim: Detection of synthetic images by masked spectrum similarity analysis. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 3855–3865, 2024.

[104] Maxime Oquab, Timothée Darcet, Théo Moutakanni, Huy Vo, Marc Szafraniec, Vasil Khalidov, Pierre Fernandez, Daniel Haziza, Francisco Massa, Alaaeldin El-Nouby, Mahmoud Assran, Nicolas Ballas, Wojciech Galuba, Russell Howes, Po-Yao Huang, Shang-Wen Li, Ishan Misra, Michael Rabbat, Vasu Sharma, Gabriel Synnaeve, Hu Xu, Hervé Jegou, Julien Mairal, Patrick Labatut, Armand Joulin, and Piotr Bojanowski. Dinov2: Learning robust visual features without supervision, 2024.

[105] Pinecone Systems, Inc. What is a vector database? `https://www.pinecone.io/learn/vector-database/`, 2025. Accessed April 20, 2025.

[106] Chroma Documentation Team. Introduction to chroma. `https://docs.trychroma.com/docs/overview/introduction`, 2025. Accessed April 20, 2025.

[107] Dhiraj Nallapaneni. Solana vs. ethereum: Investor's guide 2025. `https://coinledger.io/tools/solana-vs-ethereum`, 2025. Accessed April 20, 2025.

[108] Debbie S. Ma, Joshua Correll, and Bernd Wittenbrink. The Chicago Face Database: A Free Stimulus Set of Faces and Norming Data. *Behavior Research Methods*, 47:1122–1135, 2015.

[109] Debbie S. Ma, Justin Kantner, and Bernd Wittenbrink. Chicago Face Database: Multiracial Expansion. *Behavior Research Methods*, 2020.

[110] Pooja Lakshmi, Bernd Wittenbrink, Joshua Correll, and Debbie S. Ma. The india face set: International and cultural boundaries impact face impressions and perceptions of category membership. *Frontiers in Psychology*, 12:161, 2020.

[111] Felix Birdy. Cifake: Real and ai-generated synthetic images. `https://www.kaggle.com/datasets/birdy654/cifake-real-and-ai-generated-synthetic-images`, 2023. Accessed: 2025-07-02.

[112] Xin Huang. 140k real and fake faces. `https://www.kaggle.com/datasets/xhlulu/140k-real-and-fake-faces`, 2021. Accessed: 2025-07-02.