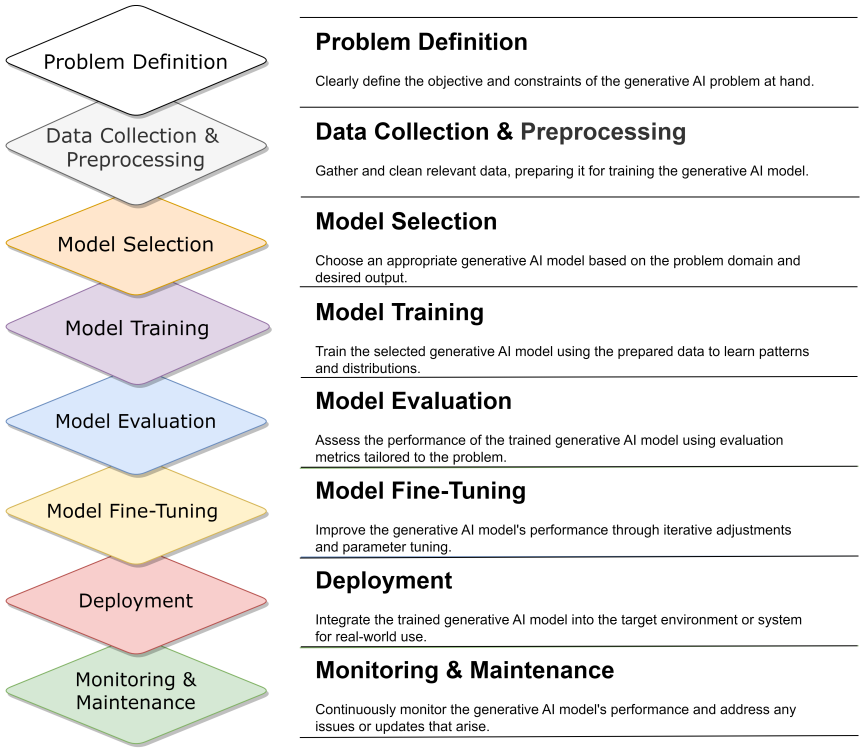

Figure 2. Implementation phases of generative AI.

The initial phase revolves around clearly defining the problem that the generative AI model aims to address. This encompasses identifying the desired outcomes, data requirements, and any constraints or limitations, thus establishing a solid foundation for subsequent stages. Accurate problem definition facilitates targeted data collection and effective model selection, streamlining the overall implementation process. The subsequent phase focuses on data collection, which entails gathering a large and representative dataset that encapsulates the patterns and characteristics that the generative model intends to learn using appropriate devices for capturing data, such as web scraping tools [33–35], microphones, cameras [36], or sensors. There are various datasets from researchers to facilitate research and benchmarking for various types of data: images—MSCOCO [37], Flickr [38]; text—Colossal Clean Crawled Corpus (C4) [39]; or audio—FSD50K [40], AudioCaps [41]. Data should be diverse and comprehensive to capture the underlying patterns effectively. Following data collection, the model selection phase begins, where the most suitable generative model architecture is chosen. This phase involves considering popular options such as VAEs, GANs, transformers, or diffusion models. The selection of an appropriate model architecture aligns with the problem requirements and paves the way for subsequent steps.

Once the generative model architecture is determined, the model training phase commences. This stage involves training the selected model using the collected or available dataset. Through this process, the model learns the underlying patterns and statistical relationships within the data. Training generative models frequently necessitates substantial computing resources, particularly for large-scale datasets and sophisticated models. To expedite training, high-performance hardware like graphics processing units (GPUs) [42] or tensor processing units (TPUs) [43] are typically employed. The specific training algorithm varies depending on the chosen model, with GANs, for instance, training a generator network to produce realistic samples while concurrently training a discriminator network to differentiate between real and generated samples [44]. Hyperparameter tuning constitutes a significant aspect of the model training phase. Various hyperparameters, including

learning rate, batch size, network architecture, and regularization techniques, influence the behavior and performance of the generative model [45]. Fine-tuning these hyperparameters is crucial for optimizing the model's convergence and overall performance [33]. Once the model training is completed, the subsequent phase involves evaluating and validating its performance. Evaluation metrics are tailored to the specific task or domain. In the case of image generation, metrics such as inception score, Frechet inception distance (FID) [46], or visual inspection can be utilized to assess the quality and diversity of the generated samples. Post-processing and refinement may be necessary in certain cases to enhance the quality or adhere to specific constraints of the generated outputs. Techniques such as image smoothing, text correction, or style transfer can be employed in this phase to refine the generated samples based on domain-specific requirements. Upon successful training and validation, the generative model is ready for deployment to generate new samples. This phase involves providing an input to the model, such as a random noise vector or a partial input and obtaining an output that aligns with the distribution of the training data. Multiple samples can be generated to explore the variety and creativity exhibited by the generative model.

The rest explores the essential requirements for generative AI, focusing on hardware, software, and user aspects. The categories of AIGC requirments are shown in Table 3. When it comes to hardware, the collection of data for generative AI tasks involves leveraging cameras, microphones, sensors, and existing datasets curated by researchers for specific purposes. For the training, fine-tuning, and hyperparameter optimization stages, powerful hardware configurations like Tesla V100 16 GB, RTX 2080Ti, NVIDIA RTX 3090 with 24 GB, and TPUs are commonly employed. However, for smaller-scale models, a GTX 1060 6 GB of DDR5 can suffice. Sample generation, which is an integral part of the generative AI process, can be achieved using more basic configurations like a CPU with an i7 3.4 GHz clock speed and a GPU such as the GTX970. On the software side, various tools and frameworks play a crucial role in different phases of generative AI. Data collection and preprocessing rely on frameworks like web scraping frameworks, Pandas, Numpy, scikit-image, torch-audio, torchtext, and RDKit. Additionally, specialized tools for data acquisition, audio recording, and motion capture are employed. To train generative models effectively, deep learning frameworks, such as TensorFlow, PyTorch, scikit-learn, and SciPy, provide comprehensive support for various model architectures and optimization algorithms. These frameworks are also instrumental in evaluating and validating the models. Furthermore, post-processing and model refinement can be facilitated using libraries like OpenCV-Python and NLTK. By understanding and fulfilling these hardware and software requirements, researchers and practitioners are well-equipped to delve into generative AI research and development. These requirements lay the foundation for creating sophisticated and high-quality generative models, enabling advancements in this exciting field of artificial intelligence.

User experience requirements for generative AI models described in Table 3 are critical in ensuring user satisfaction and successful outcomes. High-quality and realistic outputs are expected, along with customization and control options to align the generated content with user preferences. Diversity and novelty in outputs, as well as performance and efficiency, are important considerations. Interactivity and responsiveness to user input, along with ethical considerations, such as fairness and data privacy, are significant requirements. Seamless integration with existing systems and compatibility with programming languages are also valued for easy adoption. By addressing these requirements, developers and researchers can create generative AI models that meet user expectations and deliver enhanced experiences.

Table 3. AIGC requirement categories.

Category	Description
Hardware requirements	Data can be collected using cameras [36], microphones, sensors and can use datasets that are released by researchers for specific tasks [37–39]. To train, fine-tune and optimize hyperparameters—Tesla V100 16 GB [47], RTX 2080Ti [48], NVIDIA RTX 2080Ti, NVIDIA GeForce RTX 3090 with 24 GB [49], TPU [50], etc., are generally used, while GTX 1060 6 GB of DDR5 [51] can also be used to train a small-scale model. Sample generation can be performed on basic configuration like CPU—i7 3.4 GHz and GPU—GTX970 [52] or the configuration required by the generative model.
Software requirements	For data collection and preprocessing, tools such as Web scraping frameworks [33–35], Pandas [48,52,53], Numpy [53–55], scikit-image [48,55,56], torch-audio, torchtext [48], RDKit [57], data acquisition tools, audio recording software, motion capture software. To train the models, deep learning frameworks like TensorFlow [52,58,59], PyTorch [60–62], scikit-learn [52,60,63], SciPy [53,63] are used, which provide support for various generative model architectures and optimization algorithms. PyTorch [64], TensorFlow [65], scikit-learn [60]: these libraries are also used to evaluate and validate the model. For post-processing and refinement of models, libraries like opencv_python [55,66], NLTK [59,67] are used.
User-experience requirements	Key considerations for user aspects are high quality, accuracy [68] and realistic outputs [69], customization and control, diversity [70] and novelty, performance and efficiency, interactivity and responsiveness to user input, ethics [71] and data privacy [72,73], and seamless integration with existing systems.

3.2. Classification of Generative AI Models

Generative AI Model Architecture: This is the model's basic structure or design. It includes how its layers or neural networks and components are arranged and organized. The model's architecture determines how it processes and generates information, which makes it a critical aspect of its functionality and suitable for specific tasks. Table 4 describes the architecture components and training methods that are used in the generative AI models.

The classification of generative models based on architecture provides insights into the specific components and training methods that define each model as shown in Figure 3. These architectural choices have significant implications for how the models generate new data points and learn from the available data. By understanding these distinctions, researchers and practitioners can choose the most suitable generative model for their specific task or explore hybrid approaches that combine different models to leverage their respective strengths. Variational autoencoders (VAEs) have an encoder–decoder architecture and use variational inference for training. They learn compressed representations of input data and generate new samples by sampling from the learned latent space. Generative adversarial networks (GANs) consist of a generator and a discriminator. They are trained adversarially, with the generator generating synthetic samples to fool the discriminator. GANs excel at generating realistic and diverse data. Diffusion models involve a noising step followed by a denoising step. They iteratively refine noisy inputs to generate high-quality samples. Training involves learning the dynamics of the diffusion process. Transformers employ an encoder–decoder architecture and utilize self-attention mechanisms for capturing global dependencies. They are commonly used in tasks like machine translation and generate coherent sequences through supervised training. Language models, often based on recurrent neural networks (RNNs), generate sequences by predicting the next token. They are trained through supervised learning and excel at generating natural language sequences. Normalizing flow models use coupling layers to transform data while preserving density. They learn complex distributions by transforming a simple base distribution, trained via maximum-likelihood estimation. Hybrid models combine different architectures and training methods to leverage their respective strengths. They offer flexibility and tailored generative capabilities by integrating elements from multiple models.

Table 4. Architecture components and training methods used in generative AI models.

Model	Architecture Components	Training Method
Variational Autoencoders	Encoder–Decoder	Variational Inference [19]
Generative Adversarial Networks	Generator–Discriminator	Adversarial [44]
Diffusion Models	Noising (Forward)–Denoising	Iterative Refinement [31]
Transformers	Encoder–Decoder	Supervised [74]
Language Models	Recurrent Neural Networks	Supervised [75]
Normalizing Flow Models	Coupling Layers	Maximum-Likelihood Estimation [76]
Hybrid Models	Combination of Different Models	Varied

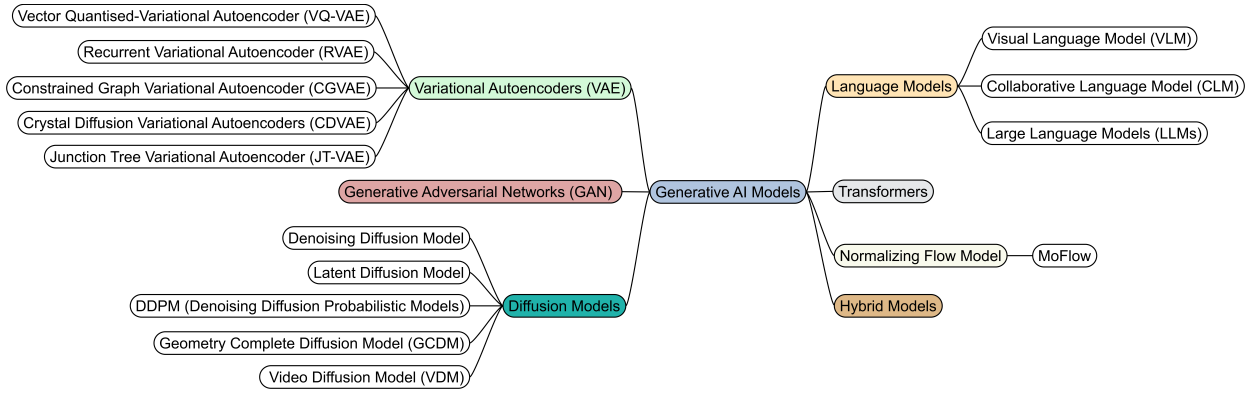

Figure 3. Classification of the generative AI models based on the architecture.

3.2.1. Variational Autoencoders (VAE)

A variational autoencoder (VAE) is a type of autoencoder that combines variational inference with an encoder–decoder architecture. Autoencoders consist of an encoder network that maps high-dimensional data to a low-dimensional representation and a decoder network that reconstructs the original input from the representation [19]. However, traditional autoencoders lack the ability to generate new data points.

In Figure 4, in a VAE, the encoder network maps the input data (x) to the parameters of a probability distribution in a latent space (z) using input layer and hidden layer composed of neural network units, such as dense or convolutional layers. This distribution is often modeled as a multivariate Gaussian with mean and covariance parameters [27] achieved in mean, variance layers. Samples are drawn from this latent space distribution in sampling layer, generated by the encoder, to produce new data points using the decoder network (y) with hidden and output layers. By sampling from the approximate posterior distribution in the latent space, VAEs can generate diverse outputs resembling the training data.

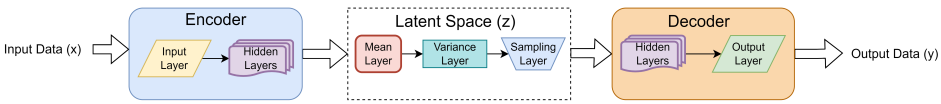

Figure 4. Typical structure of variational autoencoder (VAE).

Neural networks, such as fully connected networks or convolutional neural networks (CNNs), are commonly used as encoders and decoders in VAEs. The specific architecture

depends on the data and its complexity. For images or grid-like data, CNNs or deconvolutional neural networks (also known as convolutional transpose networks) are often employed as decoders. Training a VAE involves optimizing the model parameters by minimizing a loss function comprising a reconstruction loss and a regularization term. The reconstruction loss measures the discrepancy between the input data and the reconstructed output, while the regularization term computes the Kullback–Leibler (KL) divergence between the approximate posterior distribution and a chosen prior distribution in the latent space. This term promotes smoothness and regularization. The training process of a VAE includes selecting the network architecture, defining the loss function, and iterating through batches of input data. The encoder processes the data, latent space points are sampled, and the decoder reconstructs the input. The total loss, combining the reconstruction loss and regularization term, is computed, and gradients are used to update the model parameters through backpropagation.

While VAEs offer generative modeling and capture complex latent representations, they may suffer from issues such as blurry reconstructions and challenges in evaluating the quality of generated samples. Researchers have proposed various improvements to VAEs to address these concerns, such as vector-quantized variational autoencoder (VQ-VAE) [77], which introduces a discrete latent space by quantizing encoder outputs, leading to a more structured and interpretable latent representation; recurrent variational autoencoder (RVAE) [78] to sequential data by incorporating recurrent architectures, allowing for sequence generation and anomaly detection; constrained graph variational autoencoder (CGVAE) [79] models graph-structured data using graph neural networks, enabling generation while preserving structural properties; crystal diffusion variational autoencoders (CDVAE) [80], generating crystal structures in materials science, combining VAE with a diffusion process to learn a continuous representation of crystal structures; junction tree variational autoencoder (JT-VAE) [57], leveraging junction trees, a type of graphical model, to capture complex dependencies between variables in structured domains like natural language processing or bioinformatics.

3.2.2. Normalizing Flow Models

Normalizing flow models are deterministic and invertible transformations between the raw data space and the latent space [21]. Unlike other generative models such as GANs or VAEs, which introduce latent variables and transform them to generate new content, normalizing flow models directly solve the mapping transformation between two distributions by manipulating the Jacobian determinant [27]. In Figure 5, normalizing flow applies a sequence of invertible transformations to a simple probability distribution (z) to model more complex probability distributions using an affine coupling layer in the encoder (flow). The decoding (inverse) function is designed to be the exact inverse of the encoding function using same affine coupling layers and quick to calculate, giving normalizing flows the property of tractability [29].

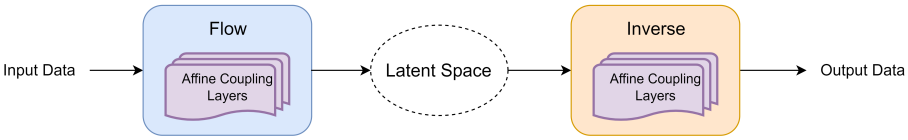

Figure 5. Typical structure of normalizing flow model.

Coupling layers play a crucial role in normalizing flow models. They are used to perform reversible transformations on the input data and latent variables. Affine coupling transformations, a specific type of coupling layer, are commonly used in normalizing flows. These transformations model complex relationships between variables while maintaining invertibility. By using element-wise multiplication and addition, the Jacobian determinant can be efficiently computed. In a coupling layer, the input data are split into fixed and transformed parts. The fixed part is typically passed through unchanged, while the