40º Simpósio Brasileiro de Bancos de Dados

40th Brazilian Symposium on Databases

SBBD 2025

Tópicos em Gerenciamento de Dados e Informações: Minicursos do SBBD 2025

Topics in Data Management and Information: Short courses of SBBD 2025

De 29 de setembro a 2 de outubro de 2025

September 29th to October 2nd, 2025

Execução Realização

UNIVERSIDADE FEDERAL DO CEARÁ

SBC Sociedade Brasileira de Computação

Cebd SBC Comissão Especial de Banco de Dados

Organização do SBBD 2025

SBBD 2025 Organization

Program Chair
Carlos Eduardo Santos Pires (UFCG, Brasil)

Short Papers Chair
Elaine Parros M. Souza (USP, Brasil)

Demos and Applications Chair
Luiz Celso Gomes Jr (UTFPR, Brasil)

WTDBD Chair
Maria Camila Nardini Barioni (UFU, Brasil)

CTDBD Chair
Valeria Cesário Times (UFPE, Brasil)

Short Courses Chair
Denio Duarte (UFFS, Brasil)

Tutorials Chair
Eduardo Cunha de Almeida (UFPR, Brasil)

Workshops Chair
Leonardo Andrade Ribeiro (UFG, Brasil)

WTAG Chair
Alessandreia Marta de Oliveira (UFJF, Brasil)

Proceedings Chair
Michele A. Brandão (UFMG, Brasil)

Marathon Chair
Arlino H. M. de Araújo (UFPI, Brasil)

Organização Geral

General Organization

Chairs

Ticiana L. Coelho da Silva (UFC, Brasil)

Regis Pires Magalhães (UFC, Brasil)

Lívia Almada Cruz (UFC, Brasil)

José Antônio Fernandes de Macêdo (UFC, Brasil)

Dados Internacionais de Catalogação na Publicação (CIP)

S612 Simpósio Brasileiro de Banco de Dados (40. : 29 set. – 02 out. 2025 : Ceará)

Minicursos do SBBD 2025 [recurso eletrônico] / organização: Michele A. Brandão. Dados eletrônicos. – Porto Alegre: Sociedade Brasileira de Computação, 2025.

67 p. : il. : PDF ; 4,3 MB

Modo de acesso: World Wide Web.
Inclui bibliografia
ISBN 978-85-7669-656-8 (e-book)

1. Computação – Brasil – Simpósio. 2. Banco de Dados. I. Brandão, Michele A. II. Sociedade Brasileira de Computação. III. Título.

CDU 004.6(063)

Ficha catalográfica elaborada por Annie Casali – CRB-10/2339

Biblioteca Digital da SBC – SBC OpenLib

Sociedade Brasileira de Computação
Av. Bento Gonçalves, 9500
Setor 4 | Prédio 43.412 | Sala 219 | Bairro
Agronomia Caixa Postal 15012 | CEP 91501-970
Porto Alegre - RS
Fone: (51) 99252-6018
sbc@sbc.org.br

O presente livro inclui dois capítulos escritos pelos autores dos minicursos selecionados e apresentados durante o XL Simpósio Brasileiro de Bancos de Dados (SBBD 2025), realizado de 29 de setembro a 2 de outubro de 2025. Os minicursos têm como objetivo apresentar temas relevantes relacionados à área de Banco de Dados e promover discussões sobre os fundamentos, tendências e desafios dos temas abordados. Cada minicurso tem quatro horas de duração e constitui uma excelente oportunidade de atualização para acadêmicos e profissionais que participam do evento. A qualidade desta edição é devida essencialmente aos autores e revisores dos trabalhos submetidos. Expressamos nossos agradecimentos pelas contribuições e discussões durante o SBBD 2025.

Os capítulos abordam conteúdos relacionados a Modelos de Difusão para geração de dados e Agentes Generativos para acesso inteligente a dados. O comitê de programa de minicursos foi composto pelos professores Denio Duarte (UFFS), Felipe Timbó (UFC), Geomar Schreiner (UFFS) e Iago Chaves (UFC), sob coordenação do primeiro.

A qualidade dessa edição é devida essencialmente aos autores e revisores dos trabalhos submetidos. Expressamos nossos fortes agradecimentos pelas contribuições e discussões durante o SBBD 2025.

Para mais informações sobre o SBBD 2025, visite https://sbbd.org.br/2025, o site desta edição do evento.

This book includes two chapters written by the authors of the selected short courses presented during the 40th Brazilian Symposium on Databases (SBBD 2025), held from September 29 to October 2, 2025. They aim to present relevant topics related to Databases. Moreover, they promote discussions on the topics' fundamentals, trends, and challenges. Each short course lasts four hours and is an excellent opportunity to update academics and professionals participating in the event. Each short course lasts four hours and is an excellent opportunity for academics and professionals participating in the event to update their knowledge.

The chapters cover content related to Diffusion models and Generative agents. The short course program committee was composed of Denio Duarte (UFFS), Felipe Timbó (UFC), Geomar Schreiner (UFFS) e Iago Chaves (UFC), under the coordination of the former.

The richness of this issue can be mainly credited to the authors and reviewers. We greatly thank them for their insightful contributions and discussions during SBBD 2025.

You can find more information about SBBD 2025, visiting the https://sbbd.org.br/2025 website of the event.

Dênio Duarte
(UFFS)

Execução
UNIVERSIDADE FEDERAL DO CEARÁ
Realização
SBC Sociedade Brasileira de Computação
Cebd SBC Comissão Especial de Banco de Dados

SUMÁRIO DOS MINICURSOS

Chapter 1

Diffusion Models: An Accessible Introduction to the State of the Art in Data Generation

Samir Braga Chaves, José Antônio Fernandes de Macêdo, Regis Pires Magalhães, Vaux Sandino Diniz Gomes, César Lincoln Cavalcante Mattos

Abstract

Generative models have received significant attention in the artificial intelligence community and are expanding their reach beyond the technical sphere. Among the most recent and promising approaches are Diffusion Models, which adopt an innovative generative modeling strategy: they progressively add noise of varying intensities to the data—a technique inspired by stochastic dynamics and known as diffusion—and then learn to reverse these perturbations. Starting from a pure noise sample, these models can generate new images, text, videos, and various other types of data with remarkably high quality, outperforming previous state-of-the-art methods in many of these tasks. This minicourse provides a formal and accessible introduction to the foundations of diffusion models, covering their underlying probabilistic principles—such as Kullback-Leibler divergence, Langevin dynamics, and score matching—as well as visual intuitions to aid comprehension. Then, it presents an overview of the current state of the art in the field, including conditioning techniques and latent diffusion models. Finally, the course discusses the research opportunities and applications that this rapidly evolving area has to offer.

1.1. Introduction

Generative models are currently receiving significant attention within the artificial intelligence community and expanding their impact beyond technical domains. They have enabled the creation of rich, personalized content—such as images, language, and music—with applications ranging from generating realistic photographs and physics-consistent animations to constructing digital worlds for gaming. Their main appeal lies in their ability to handle creative tasks, generate synthetic data, and open new scientific and artistic frontiers.

Various architectures have been proposed for data generation, each with distinct characteristics and modeling strategies. Among the most prominent are Diffusion Models (DMs) [Sohl-Dickstein et al. 2015, Song and Ermon 2019, Ho et al. 2020], which have

gained attention for their exceptional sample quality. These models generate data by gradually adding random noise through a process called *diffusion*, and then learning to reverse this corruption. By training a model to denoise, DMs can turn Gaussian noise into realistic samples of images, text, or video [Dhariwal and Nichol 2021a, Gong et al. 2022, Luo et al. 2023]. Due to their solid theoretical foundations and empirical success, DMs have joined the forefront of generative modeling.

Another widely known method is the Generative Adversarial Network (GAN) [Goodfellow et al. 2014], which employs two neural networks (a generator and a discriminator) in a competitive game to improve the quality of generated samples. Variational Autoencoder (VAE) [Kingma and Welling 2022] is an important family that learns a latent probabilistic representation of the data and uses it for generation. Normalizing Flows [Dinh et al. 2015] also rely on latent spaces, but in contrast to VAEs, they use a sequence of invertible transformations to map a simple distribution (e.g., Gaussian) into a complex one that models the data.

Both VAEs and Normalizing Flows use latent spaces as compact representations of data. VAEs associate each data point with a distribution in latent space, promoting diversity in generation, while Flows use deterministic mappings, enabling exact likelihood computation and sharper outputs. Diffusion Models provide a complementary approach: rather than relying on explicit encoder-decoder structures or invertible transformations, they define a generative process by reversing stochastic noise corruption, offering flexibility, robustness, and strong multimodal performance.

Despite their practical success, the variety of formulations and architectures in diffusion models presents challenges for those seeking a clear and unified understanding. This diversity hinders systematic comparison and model selection. Therefore, a comprehensive course is essential to consolidate existing knowledge, explain the mathematical foundations, and provide practical examples to guide implementation.

Along this minicourse, we expect the reader to be familiar with a few basic concepts. From probability and statistics, we expect a good notion of random variables, probability density functions, conditional and marginal probabilities, Bayes' Rule, and the Gaussian distribution. From deep learning, we expect a basic understanding of neural networks, how they are trained, and how they are used for inference. From calculus, the expected background is the gradient operator and the notion of vector fields.

This document presents the background necessary to understand Diffusion Models. Section 1.2 covers mathematical foundations, including latent variable models and variational inference. Section 1.3 introduces Diffusion Models, their forward and reverse processes, and the noise-based generative intuition. Section 1.4 discusses score-based models and Langevin dynamics. Section 1.5 describes common denoising architectures. Section 1.6 outlines recent advances in performance and sampling efficiency. Section 1.7 surveys applications in domains such as image, video, and text generation, bioinformatics, and tabular data. Finally, Section 1.8 summarizes key takeaways and future directions.

1.1.1. Code and resources

The implementation code and supplementary materials for this minicourse are available at the following repository: `https://github.com/InsightLab/diffusion_models_course`. There, readers will find a collection of Jupyter notebooks containing both from-scratch implementations and practical examples using the `diffusers`[1] library developed by Hugging Face. In addition to the main minicourse content, the repository also includes curated references to external implementations, papers, and hands-on examples covering a variety of use cases. This resource is intended to support further exploration, experimentation, and application of Diffusion Models in real-world scenarios.

1.2. Background

1.2.1. Generative modeling

Generative models, such as Diffusion Models, assume that all data come from a joint probability distribution of data $p(\mathbf{x})$. The two main steps of generative modeling are estimating that distribution and sampling from it. The first step consists of training a model to maximize the expected log likelihood of $p(\mathbf{x})$ or trying to minimize some divergence metric between the real distribution of the data $p(\mathbf{x})$ and the distribution $p_\theta(\mathbf{x})$ learned by the model, where θ corresponds to the model's parameters. Alternatively, one could also estimate some quantity related to the data distribution, such as the score function $\nabla \log p(\mathbf{x})$. The second step is to sample points from the learned distribution; we expect these points to be similar to those from the original data distribution. Figure 1.1 illustrates the overall idea of generative modeling.

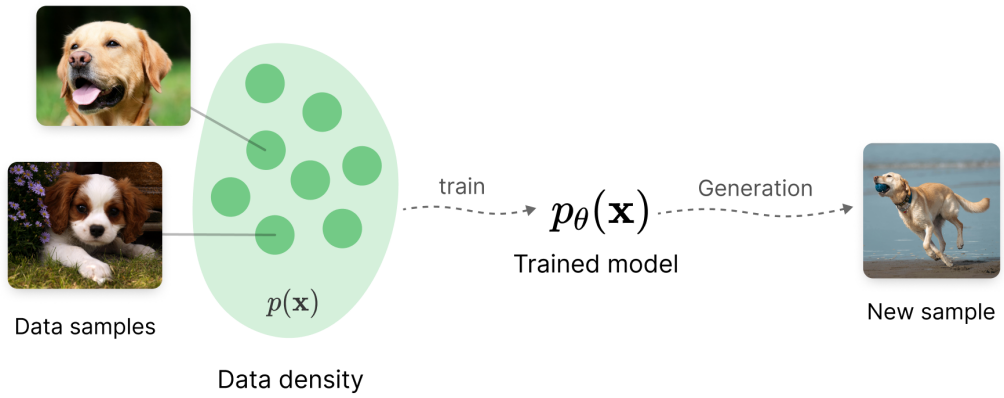


Figure 1.1. Illustration of a generative modeling process. Real data samples are used to estimate a data distribution $p_\theta(\mathbf{x})$ through training. Once trained, the model can generate new samples by sampling from $p_\theta(\mathbf{x})$, ideally matching the characteristics of the original data distribution $p(\mathbf{x})$.

1.2.2. Latent variable models

Generative models often aim to describe complex data distributions by introducing additional variables that are not directly observed, called *latent variables*. Instead of modeling the data distribution $p(\mathbf{x})$ directly, we assume that the observed data $\mathbf{x}$ is generated from

[1] `https://huggingface.co/docs/diffusers/index`

some unobserved latent representation $\mathbf{z}$. The model defines two components: a prior distribution $p(\mathbf{z})$, usually chosen to be simple (e.g., a standard Gaussian), and a likelihood model $p(\mathbf{x} \mid \mathbf{z})$ that specifies how data is generated from its latent representation.

This modeling assumption leads to the following joint distribution $p(\mathbf{x}, \mathbf{z}) = p(\mathbf{z}) \cdot p(\mathbf{x} \mid \mathbf{z})$, and from it, we can derive the posterior:

$$\underbrace{p(\mathbf{z} \mid \mathbf{x})}_{\text{posterior}} = \frac{\overbrace{p(\mathbf{x} \mid \mathbf{z})}^{\text{likelihood}} \cdot \overbrace{p(\mathbf{z})}^{\text{prior}}}{\underbrace{p(\mathbf{x})}_{\text{evidence}}}. \tag{1}$$

Here, the latent variable $\mathbf{z}$ captures the hidden structure or semantics that explain the variability in the observed data $\mathbf{x}$. The generation process is a two-step sampling procedure: first sample $\mathbf{z} \sim p(\mathbf{z})$, then sample $\mathbf{x} \sim p(\mathbf{x} \mid \mathbf{z})$. This framework is visualized in Figure 1.2.

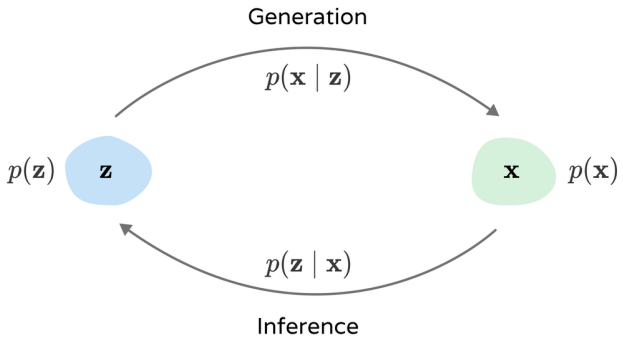


Figure 1.2. Graphical representation of a latent variable model. The top arrow illustrates the generation process: latent variables $\mathbf{z}$ are sampled from a simple prior distribution $p(\mathbf{z})$ and transformed into observed data $\mathbf{x}$ via the likelihood $p(\mathbf{x} \mid \mathbf{z})$. The bottom arrow represents inference: given observed data $\mathbf{x}$, the model estimates the posterior distribution $p(\mathbf{z} \mid \mathbf{x})$, which describes plausible latent causes for the observation.

A classic example of a latent variable model is *Principal Component Analysis* (PCA). In PCA, each data point $\mathbf{x}$ is assumed to be generated by a low-dimensional latent vector $\mathbf{z}$, linearly mapped to the high-dimensional data space by a learned projection matrix. While PCA uses linear transformations, modern approaches, such as Variational Autoencoders, extend this framework to nonlinear mappings using neural networks. Figure 1.3 can be reinterpreted here to show how latent variables mediate between prior assumptions and observed data.

One of the main computational challenges in latent variable models is the evaluation of the marginal likelihood $p(\mathbf{x})$, given by:

$$p(\mathbf{x}) = \int p(\mathbf{x} \mid \mathbf{z})\, p(\mathbf{z})\, d\mathbf{z},$$

which is often intractable in high-dimensional settings. To address this, various approximation techniques are used, such as *variational inference*, which introduces a surrogate distribution to estimate the posterior $p(\mathbf{z} \mid \mathbf{x})$.

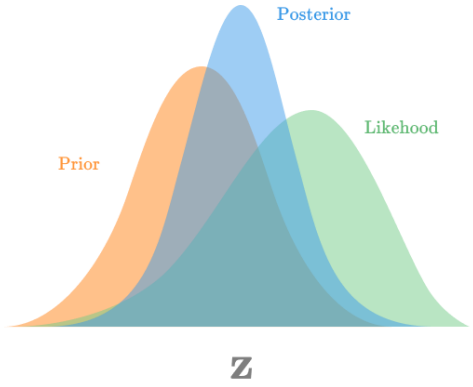


Figure 1.3. Posterior distribution $p(\mathbf{z} \mid \mathbf{x})$ (blue) as the result of combining the prior $p(\mathbf{z})$ (orange) with the likelihood $p(\mathbf{x} \mid \mathbf{z})$ (green) in a latent variable model. The z-axis represents latent factors that explain the observed data x.

In this generative framework, once the model has been trained, new data can be generated by first sampling a latent code $\mathbf{z}$ from the prior $p(\mathbf{z})$, and then generating the corresponding data point $\mathbf{x}$ using the learned decoder $p(\mathbf{x} \mid \mathbf{z})$. This approach enables controllable and interpretable generation, as the latent space can encode meaningful attributes of the data.

1.2.3. Variational Inference

In the context of latent variable models, the posterior $p(\mathbf{z} \mid \mathbf{x})$ is typically intractable to compute. To address this, one can use Variational Inference [Jordan et al. 1998, Wainwright and Jordan 2008] to approximate it with a surrogate distribution $q(\mathbf{z} \mid \mathbf{x})$. To quantify how "close" this approximation is to the true posterior, we use the Kullback-Leibler (KL) divergence: $D_{\mathrm{KL}}(q(\mathbf{z} \mid \mathbf{x}) \parallel p(\mathbf{z} \mid \mathbf{x}))$. This divergence is zero when the two distributions are identical and increases as they differ. The objective is thus to find the distribution q that minimizes the KL divergence:

$$\arg\min_{q} D_{\mathrm{KL}}(q(\mathbf{z} \mid \mathbf{x}) \parallel p(\mathbf{z} \mid \mathbf{x})). \tag{2}$$

One might question how it is possible to compute the divergence between an unknown distribution $p(\mathbf{z} \mid \mathbf{x})$ and a distribution $q(\mathbf{z} \mid \mathbf{x})$ we are trying to optimize. This is a valid concern. However, when we expand the KL divergence analytically, a useful identity arises:

$$D_{\mathrm{KL}}(q(\mathbf{z} \mid \mathbf{x}) \parallel p(\mathbf{z} \mid \mathbf{x})) = \log p(\mathbf{x}) - \underbrace{\mathbb{E}_{q(\mathbf{z}|\mathbf{x})}[\log p(\mathbf{x} \mid \mathbf{z})] - D_{\mathrm{KL}}(q(\mathbf{z} \mid \mathbf{x}) \parallel p(\mathbf{z}))}_{\text{ELBO}} \tag{3}$$

$$= \log p(\mathbf{x}) - \underbrace{\mathbb{E}_{q(\mathbf{z}|\mathbf{x})}\left[\log \frac{p(\mathbf{x}, \mathbf{z})}{q(\mathbf{z} \mid \mathbf{x})}\right]}_{\text{ELBO}}. \tag{4}$$

The right-hand side of Equation 3 consists of two components: the log evidence $\log p(\mathbf{x})$, and a term known as the Evidence Lower Bound (ELBO). Since we are optimizing with respect to q, the log evidence can be ignored, as it does not depend on q.

The ELBO includes three expressions: the variational distribution $q(\mathbf{z} \mid \mathbf{x})$ that we aim to optimize; the prior $p(\mathbf{z})$ over the latent variables, which we can choose; and the likelihood $p(\mathbf{x} \mid \mathbf{z})$, which defines the generative model and can also be optimized jointly. Equation 4 provides an alternative but equivalent form of the ELBO, which is particularly useful in the context of diffusion models. With this, the new optimization objective becomes:

$$\arg\min_q -\text{ELBO}(q) = \arg\max_q \text{ELBO}(q). \tag{5}$$

The acronym ELBO stands for **E**vidence **L**ower **BO**und, indicating that $\log p(\mathbf{x}) \geq \text{ELBO}(q)$, as illustrated in Figure 1.4, which justifies the goal of maximizing it.

A common approach is to choose a parameterized model $q_\theta(\mathbf{z} \mid \mathbf{x})$ for inference and another model $p_\phi(\mathbf{x} \mid \mathbf{z})$ for generation, where θ and ϕ denote the respective parameters. A widely adopted choice is to model both distributions as Gaussians $\mathcal{N}(\mu(\cdot), \Sigma(\cdot))$, where the functions $\mu(\cdot)$ and $\Sigma(\cdot)$ are implemented as neural networks. Figure 1.5 illustrates how a neural network can be used to implement the inference distribution as a Gaussian. The objective in Equation 5 can then be used to optimize both the inference and generative models, as is done in the training of Variational Autoencoders.

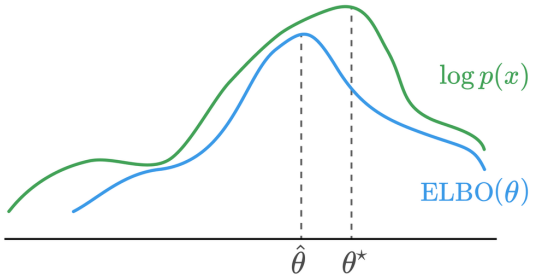
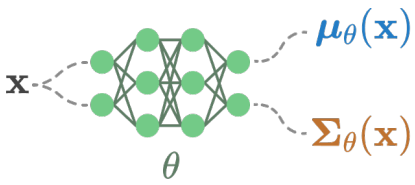


Figure 1.4. Illustration of the Evidence Lower Bound (ELBO) as a lower bound to the log-evidence $\log p(\mathbf{x})$. Maximizing the ELBO provides an approximation to the true posterior.



$$\mathcal{N}(\mathbf{z} \mid \boldsymbol{\mu}_\theta(\mathbf{x}), \boldsymbol{\Sigma}_\theta(\mathbf{x})) =: q_\theta(\mathbf{z} \mid \mathbf{x})$$

Figure 1.5. Neural network implementation of the inference model $q_\theta(\mathbf{z} \mid \mathbf{x})$ as a Gaussian distribution with learnable mean and variance.

1.2.4. History of Diffusion Models

The history of diffusion models begins with [Sohl-Dickstein et al. 2015], who proposed the idea of using a diffusion process to transform a simple prior distribution into the data distribution, where the transition kernel of this process is implemented using a neural network. This work establishes a connection between concepts from nonequilibrium thermodynamics and generative modeling.

After a four-year gap without major follow-up work, [Song and Ermon 2019] introduced a new approach to generative modeling: estimating the gradients (or score) of the data distribution and using iterative stochastic procedures (Langevin dynamics) to sample new data based on those estimated gradients. In this work, the authors employ denoising score matching [Vincent 2011] to train a neural network to estimate the score function. The method also leverages multiple noise scales to facilitate estimation—resulting in a

strategy that, while starting from different motivations, ends up resembling the original diffusion framework.

Shortly thereafter, [Ho et al. 2020] popularized modern diffusion-based generative models, demonstrating how a simple neural network can produce high-quality image generations. This work introduced a loss function based on noise prediction, which outperformed previous approaches. It also established connections between diffusion probabilistic models, denoising score matching, and Langevin dynamics, and proposed optimizing a weighted variational bound that both improves sample quality and enhances likelihood estimation.

Following that, [Song et al. 2021b] unified diffusion-based models with continuous-time stochastic processes, enabling more flexible sampling strategies (e.g., Predictor-Corrector samplers). The paper also derived a neural ordinary differential equation (ODE) capable of generating samples from the same distribution as the reverse-time stochastic differential equation (SDE), while also allowing for exact likelihood computation and improved sampling efficiency.

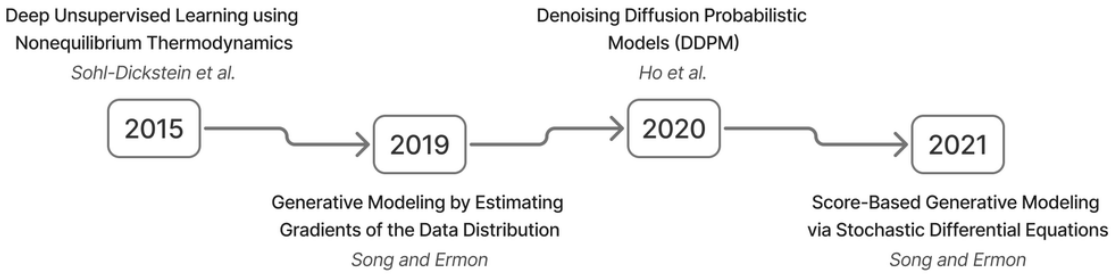


Figure 1.6. Timeline of major foundational contributions to diffusion-based generative models.

These four works constitute the foundation of the field of diffusion generative models. Figure 1.6 presents a timeline summarizing it. Since then, many other papers have focused on improving different components of the generative framework, including faster sampling [Song et al. 2021a, Lu et al. 2022, Lu et al. 2023], better conditioning [Dhariwal and Nichol 2021b, Ho and Salimans 2021, Chung et al. 2025, Tang et al. 2025], applications to inverse problems [Chung et al. 2022, Aali et al. 2023, Chung et al. 2023], and latent diffusion models [Rombach et al. 2021]. The volume of recent work in this area is extensive. A comprehensive review of these methods and their applications—ranging from computer vision, natural language processing, and time series modeling to interdisciplinary scientific domains—can be found in [Yang et al. 2024].

1.3. Denoising Diffusion Probabilistic Models

In this section, we present a formulation of diffusion models based on the work of [Ho et al. 2020], which frames diffusion models as latent variable models and employs variational inference to derive their objective function. The core idea is to progressively add noise to data (e.g., images, audio, or text) until it is transformed into pure random noise. This final state corresponds to an isotropic Gaussian distribution, where the variance is equal in all directions. The model aims to learn how to reverse this noising process. The

forward and reverse stages are illustrated in Figure 1.7. By sampling from the Gaussian prior and applying the learned reverse process, generating new data that resembles samples from the original distribution is possible.

This section focuses on the key steps and formulas, omitting most derivations to emphasize the underlying motivation and how the core processes are formulated. A more complete treatment can be found in the original paper [Ho et al. 2020] and in books such as [Prince 2023, Bishop and Bishop 2024, Murphy 2022].

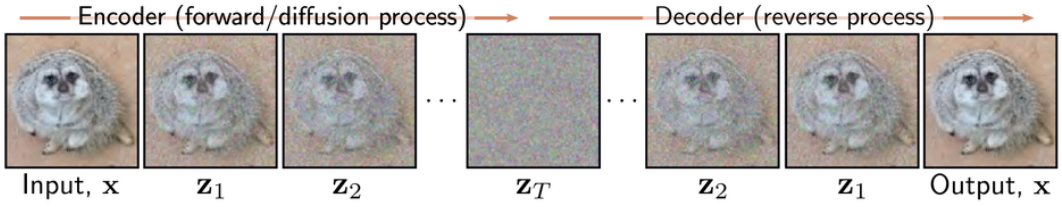


Figure 1.7. Illustration of the forward (noising) and reverse (denoising) diffusion processes used in DDPMs, starting from a data sample and ending in Gaussian noise, and vice versa. Image reproduced from the [Prince 2023].

1.3.1. Forward diffusion process

The forward diffusion process refers to the first stage of the diffusion model framework. In this step, noise is progressively added to the data until it becomes indistinguishable from pure random noise. The result is an isotropic Gaussian distribution with the same variance in all directions and zero covariance, implying that all dimensions are uncorrelated and statistically independent.

Let us define a series of discrete time steps $t = 1, 2, \ldots, T$, and let $\mathbf{x} \sim p(\mathbf{x})$ denote a sample drawn from the data distribution. For instance, $\mathbf{x}$ may be an original image from the training dataset. We introduce T latent variables $\mathbf{z}_1, \mathbf{z}_2, \ldots, \mathbf{z}_T$, where $\mathbf{z}_T \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$.

The noise is added through a Markov chain of diffusion steps, where each noisy version $\mathbf{z}_t$ depends only on the previous step $\mathbf{z}_{t-1}$. As a reminder, smaller values of t correspond to earlier steps in the process (i.e., less noise), and as $t \to T$, the distribution approaches an isotropic Gaussian. Figure 1.8 illustrates how this process unfolds for a simple 2D data distribution.

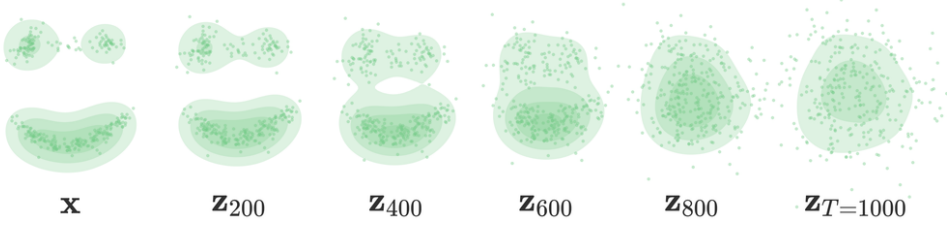


Figure 1.8. Illustration of the forward diffusion process on a 2D data distribution. The images were taken from an interactive visualization tool available at https://alechelbling.com/Diffusion-Explorer.

Then, let us define $q(\mathbf{z}_t \mid \mathbf{z}_{t-1})$ as a Gaussian distribution function conditioned on the $\mathbf{z}_{t-1}$ version of our image, which outputs the noisier $\mathbf{z}_t$ version. We also define $\beta_t \in (0,1)$ as the variance schedule controlling the amount of noise added at time step t. We define the transition at each step as a conditional Gaussian distribution:

$$q(\mathbf{z}_t \mid \mathbf{z}_{t-1}) = \mathcal{N}(\mathbf{z}_t \mid \sqrt{1-\beta_t}\mathbf{z}_{t-1}, \beta_t \mathbf{I}), \tag{6}$$

where the mean $\sqrt{1-\beta_t}\mathbf{z}_{t-1}$ is just a scaled version of the previous step $\mathbf{z}_{t-1}$ and the variance is β_t. The complete forward process is described as a sequence of conditional Gaussian distributions:

$$q(\mathbf{z}_{1:T} \mid \mathbf{x}) = q(\mathbf{z}_1 \mid \mathbf{x}) \prod_{t=2}^{T} q(\mathbf{z}_t \mid \mathbf{z}_{t-1}). \tag{7}$$

1.3.1.1. One step forward process

One key concept in the forward process is that we can directly find $\mathbf{z}_t$ given $\mathbf{x}$ without iterating over all intermediate time steps. This is only possible because the linear combination of independent Gaussian variables is also a Gaussian. Let us define $\alpha_t = 1 - \beta_t$, the cumulative product of $\alpha_{1:t}$ as $\bar{\alpha}_t = \prod_{i=1}^{t} \alpha_i$ and present the reparametrization trick bellow formula:

$$\mathcal{N}(\mu, \sigma^2) = \mu + \sigma \cdot \varepsilon \text{ , where } \varepsilon \sim \mathcal{N}(0,1). \tag{8}$$

Then, we can write Equation 6 at some t as:

$$\mathbf{z}_t = \sqrt{\bar{\alpha}_t}\mathbf{x} + \sqrt{1-\bar{\alpha}_t}\varepsilon_t, \quad \varepsilon_t \sim \mathcal{N}(\mathbf{0}, \mathbf{I}). \tag{9}$$

Finally, we can rewrite the equation in the original form:

$$q(\mathbf{z}_t \mid \mathbf{x}) = \mathcal{N}(\mathbf{z}_t \mid \sqrt{\bar{\alpha}_t}\mathbf{x}, (1-\bar{\alpha}_t)\mathbf{I}). \tag{10}$$

Note that, to get to any time step t, we only need the original sample image $\mathbf{x}$ and the cumulative noise. Figure 1.9 shows how $q(\mathbf{z}_t \mid \mathbf{x})$ evolves over time.

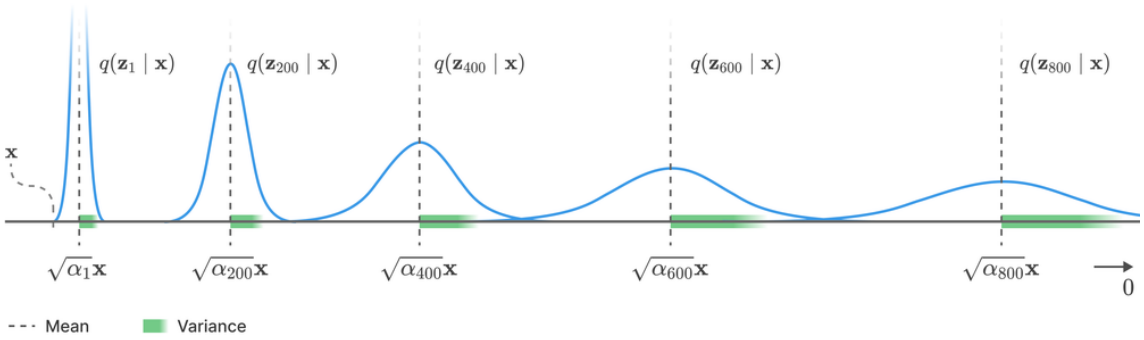


Figure 1.9. Variance evolution in the forward process as a function of time step t. As t increases, the distribution transitions from the original data to pure noise.

1.3.1.2. Noise schedule

In diffusion models, choosing the noise schedule $\{\beta_t\}_{t=1}^{T}$—which controls the variance added at each diffusion step—is a crucial design decision. The noise schedule directly influences the quality of training and sampling, as it determines how quickly the data distribution is destroyed in the forward process and how effectively the model can learn to reverse that process. Recall that each forward transition is modeled as: $q(\mathbf{z}_t \mid \mathbf{z}_{t-1}) = \mathcal{N}(\mathbf{z}_t \mid \sqrt{1-\beta_t}\mathbf{z}_{t-1}, \beta_t \mathbf{I})$ and the cumulative noise over time is encoded in $\bar{\alpha}_t = \prod_{i=1}^{t}(1-\beta_i)$.

A well-designed noise schedule adds small amounts of noise in early steps—preserving data structure and enabling meaningful denoising—and progressively increases it until the input becomes an isotropic Gaussian. The *linear schedule*, used in the original DDPM formulation [Ho et al. 2020], linearly increases β_t from a small to a large value, offering a simple and effective strategy. An alternative is the *cosine schedule*, proposed in [Nichol and Dhariwal 2021], which defines the cumulative noise $\bar{\alpha}_t$ via a cosine-shaped curve. This concentrates noise addition toward the end of the process, improving perceptual quality and training stability.

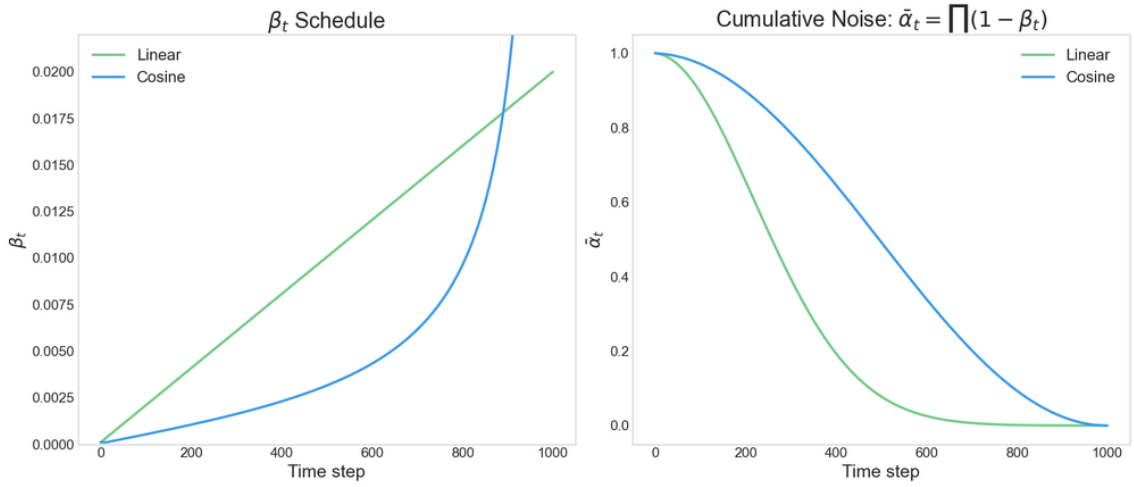


Figure 1.10. Comparison between linear and cosine noise schedules in diffusion models. The left plot shows the values of β_t over time; the right plot shows the cumulative product $\bar{\alpha}_t$, which determines how quickly the signal is destroyed.

Figure 1.10 compares linear and cosine schedules in terms of β_t and the cumulative product $\bar{\alpha}_t$ over time. While most diffusion models use predefined noise schedules, other approaches explore learning the noise schedule directly from data. For example, [Kingma et al. 2021] proposes a variational framework where the parameters of the forward process, including the variance schedule, are optimized jointly with the generative model. In summary, the choice of noise schedule affects the learning dynamics during training and plays a key role in how smoothly the model can reverse the diffusion trajectory during sampling.

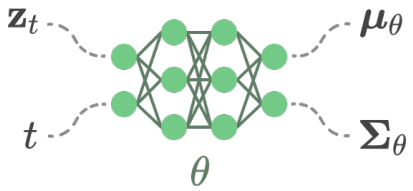


Figure 1.11. Gaussian kernel used in the reverse diffusion process to model $p_\theta(\mathbf{z}_{t-1} \mid \mathbf{z}_t)$.

1.3.2. Reverse diffusion process

In the reverse diffusion process, the goal is to sample from the distribution $q(\mathbf{z}_{t-1} \mid \mathbf{z}_t)$, given that we know how to sample from the forward distribution $q(\mathbf{z}_t \mid \mathbf{z}_{t-1})$. This makes it possible to recover a sample $\mathbf{x} \sim q(\mathbf{x})$ from the original data distribution, starting from a pure noise input $\mathbf{z}_T \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$.

A natural first attempt is to apply Bayes' theorem:

$$q(\mathbf{z}_{t-1} \mid \mathbf{z}_t) = \frac{q(\mathbf{z}_t \mid \mathbf{z}_{t-1}) q(\mathbf{z}_{t-1})}{q(\mathbf{z}_t)}. \tag{11}$$

However, as discussed in Section 1.2.2, the marginal distribution $q(\mathbf{z}_t)$ is intractable. To address this, we use variational inference and introduce a surrogate model that approximates the target distribution:

$$q(\mathbf{z}_{1:T}, \mathbf{x}) = q(\mathbf{x}) q(\mathbf{z}_1 \mid \mathbf{x}) \prod_{t=2}^{T} q(\mathbf{z}_t \mid \mathbf{z}_{t-1}) \tag{12}$$

with the surrogate defined as:

$$p_\theta(\mathbf{z}_{1:T}, \mathbf{x}) = p(\mathbf{z}_T) p_\theta(\mathbf{x} \mid \mathbf{z}_1) \prod_{t=2}^{T} p_\theta(\mathbf{z}_{t-1} \mid \mathbf{z}_t), \tag{13}$$

where the reverse transitions are modeled as Gaussian distributions:

$$p_\theta(\mathbf{z}_{t-1} \mid \mathbf{z}_t) = \mathcal{N}(\mathbf{z}_{t-1} \mid \mu_\theta(\mathbf{z}_t, t), \Sigma_\theta(\mathbf{z}_t, t)). \tag{14}$$

Here, θ represents the learnable parameters. The functions μ_θ and Σ_θ can be implemented as neural networks, as illustrated in Figure 1.11. However, as we will see, this formulation can be simplified so that a neural network does not need to model the full Gaussian parameters directly.

We now use the KL divergence to measure the similarity between the true and surrogate distributions, resulting in the following optimization problem:

$$\hat{\theta} := \arg\min_\theta D_{\mathrm{KL}}(q(\mathbf{x}, \mathbf{z}_{1:T}) \parallel p_\theta(\mathbf{x}, \mathbf{z}_{1:T})). \tag{15}$$

From Equations 4 and 5, this leads to the maximization of the Evidence Lower Bound (ELBO):

$$\hat{\theta} = \arg\max_\theta \underbrace{\mathbb{E}_{q(\mathbf{x}, \mathbf{z}_{1:T})} \left[\log \frac{p_\theta(\mathbf{x}, \mathbf{z}_{1:T})}{q(\mathbf{z}_{1:T} \mid \mathbf{x})} \right]}_{\text{ELBO}}. \tag{16}$$

This ELBO resembles Section 1.2.3, but now with T latent variables. After derivation, the resulting objective becomes:

$$\hat{\theta} = \arg\max_{\theta} \underbrace{\mathbb{E}_{q(\mathbf{x},\mathbf{z}_1)}\left[p_\theta(\mathbf{x} \mid \mathbf{z}_1)\right]}_{L_0} + \sum_{t=2}^{T} \underbrace{\mathbb{E}_{q(\mathbf{x},\mathbf{z}_t)}\left[D_{\mathrm{KL}}\left(q(\mathbf{z}_{t-1} \mid \mathbf{z}_t, \mathbf{x}) \parallel p_\theta(\mathbf{z}_{t-1} \mid \mathbf{z}_t)\right)\right]}_{L_1, L_2, \ldots, L_{T-1}}. \quad (17)$$

We can now use this objective to optimize the parameters θ of the reverse Gaussian distribution from Equation 14. The KL divergence between two multivariate Gaussians admits a closed-form solution, and each L_t term from Equation 17 becomes:

$$L_t = D_{\mathrm{KL}}\left(q(\mathbf{z}_{t-1} \mid \mathbf{z}_t, \mathbf{x}) \parallel p_\theta(\mathbf{z}_{t-1} \mid \mathbf{z}_t)\right) \quad (18)$$

$$= D_{\mathrm{KL}}(\mathcal{N}(\mathbf{z}_{t-1} \mid \tilde{\mu}(\mathbf{z}_t, \mathbf{x}), \tilde{\beta}_t \mathbf{I}) \parallel \mathcal{N}(\mathbf{z}_{t-1} \mid \mu_\theta(\mathbf{z}_t, t), \Sigma_\theta(\mathbf{z}_t, t))), \quad (19)$$

where

$$\tilde{\mu}_t(\mathbf{z}_t, \mathbf{x}) = \frac{1}{\sqrt{\alpha_t}}\left(\mathbf{z}_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}}\varepsilon_t\right), \quad \tilde{\beta}_t = \frac{1-\bar{\alpha}_{t-1}}{1-\bar{\alpha}_t} \cdot \beta_t, \quad (20)$$

and ε_t comes from the Equation 9. The equalities of Equation 20 come from the definition of the Gaussian density function, assuming the covariance matrix is diagonal $\tilde{\beta}_t \mathbf{I}$.

Algorithm 1 Algorithm for training a denoising diffusion probabilistic model, originally described in [Bishop and Bishop 2024].

Input: Training data $\mathcal{D} = \{x_n\}$, Noise schedule $\{\beta_1, \ldots, \beta_T\}$
Output: Network parameters θ
for $t \in \{1, \ldots, T\}$ **do**
    $\alpha_t \leftarrow \prod_{\tau=1}^{t}(1-\beta_\tau)$ ▷ Calculate alphas from betas
end for
repeat
    $\mathbf{x} \sim \mathcal{D}$ ▷ Sample a data point
    $t \sim \{1, \ldots, T\}$ ▷ Sample a point along the Markov chain
    $\varepsilon \sim \mathcal{N}(\varepsilon \mid \mathbf{0}, \mathbf{I})$ ▷ Sample a noise vector
    $\mathbf{z}_t \leftarrow \sqrt{\alpha_t}\mathbf{x} + \sqrt{1-\alpha_t}\,\varepsilon$ ▷ Evaluate noisy latent variable
    $\mathcal{L}(\theta) \leftarrow \|\varepsilon_\theta(\mathbf{z}_t, t) - \varepsilon\|^2$ ▷ Compute loss term
    Take optimizer step
until converged
return θ

We must learn a neural network to approximate the conditioned probability distributions in the reverse diffusion. We would like to train μ_θ to predict $\tilde{\mu}_t = \frac{1}{\sqrt{\alpha_t}}\left(\mathbf{z}_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}}\varepsilon_t\right)$. Because $\mathbf{z}_t$ is available as input at training time, we can reparameterize the Gaussian noise term instead to make it predict ε_t from the input $\mathbf{z}_t$ at time step t:

$$\mu_\theta(\mathbf{z}_t, t) = \frac{1}{\sqrt{\alpha_t}}\left(\mathbf{z}_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}}\varepsilon_\theta(\mathbf{z}_t, t)\right). \quad (21)$$

Given that, L_t becomes:

$$L_t(\theta) = \mathbb{E}_{\mathbf{x},\varepsilon}\left[\frac{1}{2\|\Sigma_\theta(\mathbf{z}_t,t)\|_2^2}\|\tilde{\mu}_t(\mathbf{z}_t,\mathbf{x}) - \mu_\theta(\mathbf{z}_t,t)\|^2\right] \tag{22}$$

$$= \mathbb{E}_{\mathbf{x},\varepsilon}\left[\frac{1}{2\|\Sigma_\theta(\mathbf{z}_t,t)\|_2^2}\left\|\frac{1}{\sqrt{\alpha_t}}\left(\mathbf{z}_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}}\varepsilon_t\right) - \frac{1}{\sqrt{\alpha_t}}\left(\mathbf{z}_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}}\varepsilon_\theta(\mathbf{z}_t,t)\right)\right\|^2\right] \tag{23}$$

$$= \mathbb{E}_{\mathbf{x},\varepsilon}\left[\frac{(1-\alpha_t)^2}{2\alpha_t(1-\bar{\alpha}_t)\|\Sigma_\theta(\mathbf{z}_t,t)\|_2^2}\|\varepsilon_t - \varepsilon_\theta(\mathbf{z}_t,t)\|^2\right]. \tag{24}$$

Empirically, [Ho et al. 2020] found that training works better when the weighting term is removed, leading to the simplified loss:

$$L_t^{\text{simple}}(\theta) = \mathbb{E}_{\mathbf{x},\varepsilon_t}\left[\|\varepsilon_t - \varepsilon_\theta(\mathbf{z}_t,t)\|^2\right] \tag{25}$$

$$= \mathbb{E}_{\mathbf{x},\varepsilon_t}\left[\|\varepsilon_t - \varepsilon_\theta(\sqrt{\bar{\alpha}_t}\mathbf{x} + \sqrt{1-\bar{\alpha}_t}\varepsilon_t, t)\|^2\right]. \tag{26}$$

Thus, the final complete objective is:

$$\mathcal{L}(\theta) := \mathbb{E}_{t\sim[1,T],\mathbf{x},\varepsilon_t}\left[\|\varepsilon_t - \varepsilon_\theta(\sqrt{\bar{\alpha}_t}\mathbf{x} + \sqrt{1-\bar{\alpha}_t}\varepsilon_t, t)\|^2\right]. \tag{27}$$

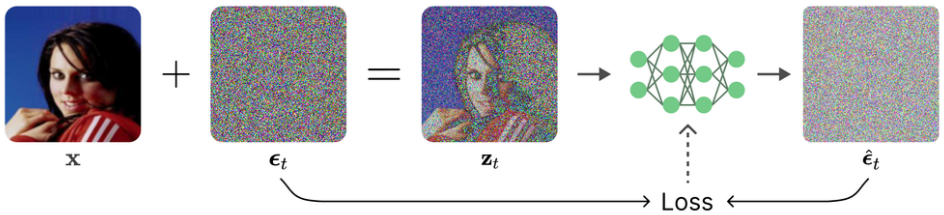


Figure 1.12. Simplified training objective used in DDPM, where the network ε_θ learns to predict the added noise ε_t given $\mathbf{z}_t$ and t.

This final training objective from Equation 25 is illustrated in Figure 1.12. Algorithm 1 brings the steps to train the diffusion model using this loss, while Algorithm 2 shows how to generate new data using the trained model ε_θ.

With the DDPM objective established, the next sections explore an alternative formulation (score-based models), practical implementations, and improvements to the base model.

1.4. Diffusion Models as score-based models

Diffusion models belong to the broader class of score-based generative models. The term *score* comes from the (Stein) score function [Cox and Hinkley 1974] that is the gradient of the log probability density function $\nabla \log p(\mathbf{x})$. The score function is a vector field pointing to places with higher data density. Estimating this function is helpful due to methods such as *stochastic gradient Langevin dynamics*, which is a Markov Chain Monte

Algorithm 2 Algorithm for sampling from a denoising diffusion probabilistic model, originally described in [Bishop and Bishop 2024].

Input: Trained denoising network $\varepsilon_\theta(\mathbf{z},t)$, Noise schedule $\{\beta_1,\ldots,\beta_T\}$
Output: Sample vector $\mathbf{x}$ in data space
$\mathbf{z}_T \sim \mathcal{N}(\mathbf{z} \mid \mathbf{0},\mathbf{I})$ ▷ Sample from final latent space
for $t \in \{T,\ldots,2\}$ **do**
    $\alpha_t \leftarrow \prod_{\tau=1}^{t}(1-\beta_\tau)$ ▷ Calculate alpha
    $\mu_\theta(\mathbf{z}_t,t) \leftarrow \frac{1}{\sqrt{1-\beta_t}}\left(\mathbf{z}_t - \frac{\beta_t}{\sqrt{1-\alpha_t}}\varepsilon_\theta(\mathbf{z}_t,t)\right)$ ▷ Evaluate network output
    $\varepsilon \sim \mathcal{N}(\varepsilon \mid \mathbf{0},\mathbf{I})$ ▷ Sample a noise vector
    $\mathbf{z}_{t-1} \leftarrow \mu_\theta(\mathbf{z}_t,t) + \sqrt{\beta_t}\,\varepsilon$ ▷ Add scaled noise
end for
$\mathbf{x} \leftarrow \frac{1}{\sqrt{1-\beta_1}}\left(\mathbf{z}_1 - \frac{\beta_1}{\sqrt{1-\alpha_1}}\varepsilon_\theta(\mathbf{z}_1,t)\right)$ ▷ Final denoising step
return $\mathbf{x}$

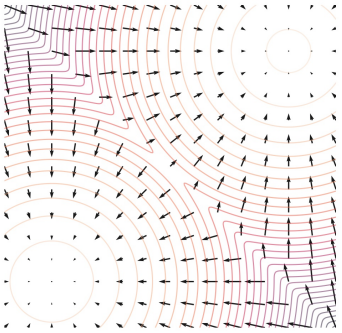


Figure 1.13. A vector field representing the score function and contours representing the density function of a mixture of two Gaussians. Source: https://yang-song.net/blog/2021/score.

Carlo (MCMC) procedure that can sample from a distribution $p(\mathbf{x})$ using only its score function. Figure 1.13 shows a representation of both score and density functions.

To estimate the score, a series of techniques called Score Matching is used. We can train a model to do so by minimizing the Fisher divergence between the model and the score of the data distribution, defined as:

$$\mathbb{E}_{p(\mathbf{x})}[\|\nabla_\mathbf{x}\log p(\mathbf{x}) - \mathbf{s}_\theta(\mathbf{x})\|_2^2]. \tag{28}$$

The Fisher divergence compares the squared distance between the ground-truth data score and the score-based model. Directly computing this divergence, however, is infeasible because it requires access to the unknown data score $\nabla_\mathbf{x}\log p(\mathbf{x})$. That is the reason we need score-matching. Many works have proposed different strategies to perform this score estimation [Hyvärinen 2005, Vincent 2011, Song et al. 2019], but here we will focus on the *denoising score matching*.

1.4.1. Denoising score matching

Denoising score matching [Vincent 2011] is a practical variation of the original score matching technique. The core idea is simple: instead of estimating the score of the orig-

inal data distribution directly (something often challenging), we first corrupt the data by adding a known amount of noise. Formally, we define a noise distribution $q(\mathbf{z} \mid \mathbf{x})$ that perturbs the original data point $\mathbf{x}$ into a noisy version $\mathbf{z}$.

By applying score matching to this noisy distribution $q(\mathbf{z})$, we aim to train a model $\mathbf{s}_\theta(\mathbf{z})$ that estimates the score of $q(\mathbf{z})$. The objective function turns out to be equivalent to minimizing the difference between the predicted score and the score of the noise distribution itself:

$$\frac{1}{2}\mathbb{E}_{q(\mathbf{z}|\mathbf{x})p_{\text{data}}(\mathbf{x})}\left[\|\mathbf{s}_\theta(\mathbf{z}) - \nabla_\mathbf{z} \log q(\mathbf{z} \mid \mathbf{x})\|_2^2\right]. \tag{29}$$

Intuitively, this means the model learns to "denoise" $\mathbf{z}$ by estimating in which direction the noise moved it from its original clean version $\mathbf{x}$. The optimal solution $\mathbf{s}_{\theta^*}$ to this objective satisfies $\mathbf{s}_{\theta^*}(\mathbf{z}) = \nabla_\mathbf{z} \log q(\mathbf{z})$ almost everywhere, as proved in [Vincent 2011]. Moreover, when the noise level is small, the perturbed distribution $q(\mathbf{z})$ closely approximates the true data distribution $p_{\text{data}}(\mathbf{x})$, so their gradients are also close $\nabla_\mathbf{z} \log q(\mathbf{z}) \approx \nabla_\mathbf{x} \log p_{\text{data}}(\mathbf{x})$.

A common choice for the noise distribution is a simple Gaussian centered at the original data point $q(\mathbf{z} \mid \mathbf{x}) = \mathcal{N}(\mathbf{z} \mid \mathbf{x}, \sigma^2 I)$, for which we can compute the score analytically:

$$\nabla_\mathbf{z} \log q(\mathbf{z} \mid \mathbf{x}) = -\frac{\mathbf{z} - \mathbf{x}}{\sigma^2}. \tag{30}$$

Plugging this into the objective, we obtain the final denoising score matching loss for a fixed noise level σ:

$$\frac{1}{2}\mathbb{E}_{\mathbf{x},\mathbf{z}\sim\mathcal{N}(\mathbf{x},\sigma^2 I)}\left[\left\|\mathbf{s}_\theta(\mathbf{z},\sigma) + \frac{\mathbf{z} - \mathbf{x}}{\sigma^2}\right\|_2^2\right]. \tag{31}$$

In this formulation, the model learns to predict the direction and intensity of the noise that was added, which effectively teaches it the gradient of the log-density of the data through supervised learning on noisy inputs.

1.4.2. Sampling with Langevin dynamics

Langevin dynamics (LD) provides a way to generate samples from a probability distribution $p(\mathbf{x})$ using only its score function, $\nabla_\mathbf{x} \log p(\mathbf{x})$. The idea is to start from a random point $\tilde{\mathbf{x}}_0$ drawn from some simple prior distribution $\pi(\mathbf{x})$, and then iteratively refine it using both the score function and added noise. Given a fixed step size $\alpha > 0$, the Langevin update rule is:

$$\tilde{\mathbf{x}}_i = \tilde{\mathbf{x}}_{i-1} + \frac{\alpha}{2}\nabla_\mathbf{x} \log p(\tilde{\mathbf{x}}_{i-1}) + \sqrt{\alpha}\,\omega_i, \quad i = 0, 1, \cdots, K, \tag{32}$$

where $\omega_t \sim \mathcal{N}(0, I)$ is Gaussian noise. Intuitively, each update step moves the sample in the direction where the data density $p(\mathbf{x})$ increases (guided by the score), while adding a small amount of random noise to explore the space. Figure 1.14 shows a visualization of Langevin dynamics sampling.

Suppose we apply this update many times (i.e., letting $K \to \infty$) and use a tiny step size ($\alpha \to 0$). In that case, the final sample $\tilde{\mathbf{x}}_K$ will converge to an actual sample from $p(\mathbf{x})$, under certain mathematical conditions. When α is small and K is large, this procedure can

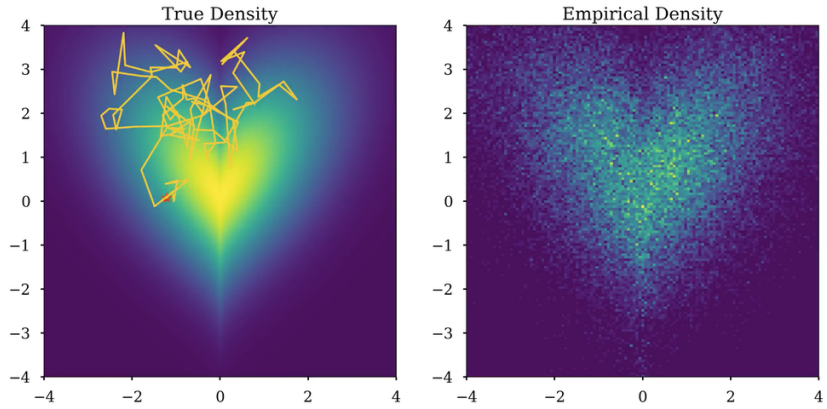


Figure 1.14. Visualization of Langevin dynamics sampling. Left: The true data density is shown as a heatmap, with yellow indicating higher probability. The orange trajectory represents a single sample path generated using LD. Right: The empirical density obtained from multiple samples, showing that the sampling process successfully approximates the true data distribution.

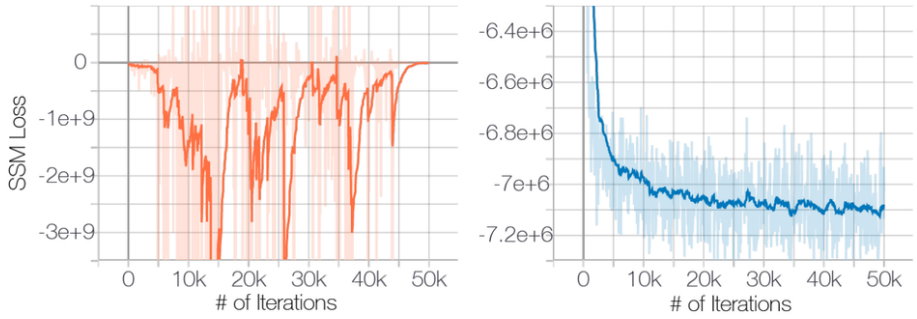


Figure 1.15. Effect of the manifold hypothesis. Left: SSM fails on unperturbed CIFAR-10. Right: Adding Gaussian noise $\mathcal{N}(0, 0.0001)$ enables convergence by giving full support over $\mathbb{R}^D$.

produce high-quality approximate samples even without applying a Metropolis-Hastings correction step, which is sometimes needed to ensure exactness.

An important observation is that the update rule above only depends on the score function $\nabla_{\mathbf{x}} \log p(\mathbf{x})$. This means that to sample from the true data distribution $p_{\text{data}}(\mathbf{x})$, we can first train a neural network score estimator $\mathbf{s}_\theta(\mathbf{x})$ to approximate this gradient. Once trained, we can use $\mathbf{s}_\theta(\mathbf{x})$ in place of the true score in Langevin dynamics to generate samples that follow $p_{\text{data}}(\mathbf{x})$. This sampling strategy is a key component of the approach known as *score-based generative modeling*.

1.4.3. Limitations of score matching and Langevin dynamics

While score-based generative modeling provides an elegant framework for sampling from complex distributions, it faces challenges when data lies on low-dimensional structures or in sparse regions.

Score undefinedness. Real-world data often lies on lower-dimensional struc-

tures in a high-dimensional space (the Manifold Hypothesis [Fefferman et al. 2013]), making the score $\nabla_{\mathbf{x}} \log p_{\text{data}}(\mathbf{x})$ undefined off the manifold. Score matching requires full support over $\mathbb{R}^D$, which is rarely satisfied. A common solution is to add slight Gaussian noise to the data, ensuring tractability (Figure 1.15).

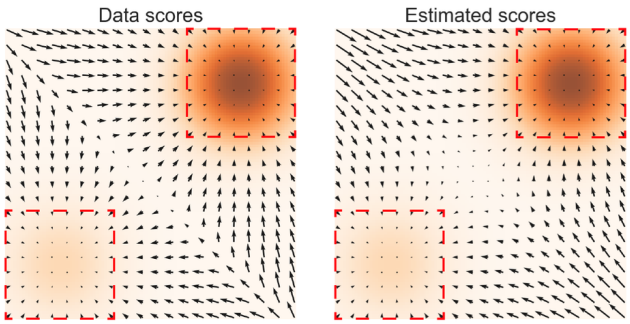


Figure 1.16. Score estimation in a Gaussian mixture. Left: Ground-truth scores. Right: Estimated scores. Accuracy is high only near the dense modes.

Low-density regions. Score matching minimizes a squared error weighted by $p_{\text{data}}(\mathbf{x})$, so errors in low-density areas contribute little:

$$\mathbb{E}_{p(\mathbf{x})}[\|\nabla_{\mathbf{x}} \log p(\mathbf{x}) - \mathbf{s}_\theta(\mathbf{x})\|_2^2] = \int p(\mathbf{x}) \|\cdot\|^2 d\mathbf{x}.$$

As a result, the model may fail to learn accurate scores in these regions, which can affect global structure (Figure 1.16).

Mode balancing failure in LD. Langevin Dynamics struggles with multimodal distributions, especially when modes are separated by low-density regions. For $p_{\text{data}} = \pi p_1 + (1-\pi) p_2$ with disjoint supports, the score is locally accurate but contains no information about π. LD cannot recover mixture proportions and requires tiny steps to bridge modes. Standard LD fails under these conditions, unlike annealed variants (Figure 1.17).

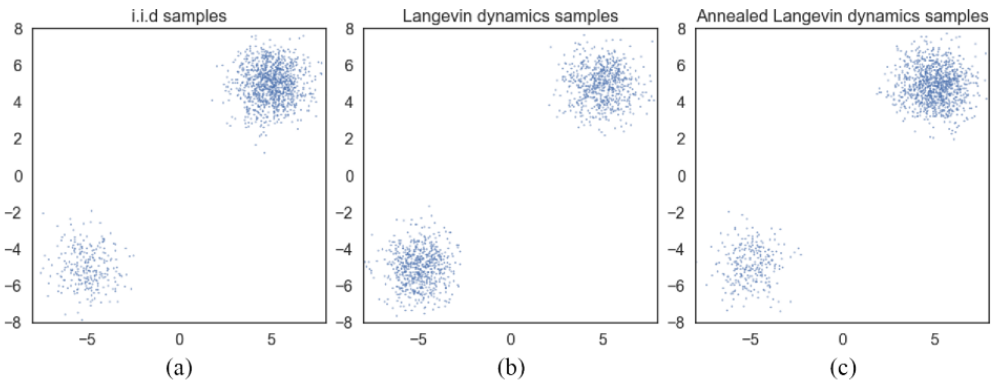


Figure 1.17. Langevin Dynamics vs. Annealed LD. LD fails to reflect correct mode proportions.

1.4.4. Multiple noise scales

One challenge in the presented strategy is choosing the right noise level for perturbing the data. Large noise values help cover low-density regions but overly distort the data, while small noise preserves details but fails to provide coverage in sparse areas.

To address this, we perturb the data with multiple noise scales simultaneously. Let $\{\sigma_1, \sigma_2, \ldots, \sigma_T\}$ be an increasing sequence of standard deviations, and define the perturbed distributions as $q(\mathbf{z}_t \mid \mathbf{x}) = \mathcal{N}(\mathbf{z}_t \mid \mathbf{x}, \sigma_t^2 \mathbf{I})$. Sampling from $q(\mathbf{z}_t \mid \mathbf{x})$ is straightforward: draw $\mathbf{x} \sim p_{\text{data}}$ and add Gaussian noise $\sigma_t \varepsilon$ with $\varepsilon \sim \mathcal{N}(0, I)$.

We then train a score network $\mathbf{s}_\theta(\mathbf{x}, \sigma)$ to estimate the score $\nabla_{\mathbf{z}_t} \log q(\mathbf{z}_t \mid \mathbf{x})$ for all noise levels σ_t, using denoising score matching:

$$L_t(\theta) = \frac{1}{2} \mathbb{E}_{\mathbf{x}, \mathbf{z}_t \sim \mathcal{N}(\mathbf{x}, \sigma_t^2 I)} \left[\left\| \mathbf{s}_\theta(\mathbf{z}_t, \sigma_t) + \frac{\mathbf{z}_t - \mathbf{x}}{\sigma_t^2} \right\|_2^2 \right].$$

The full loss combines all scales with a weighting function $w(\sigma)$:

$$\mathcal{L}(\theta) = \frac{1}{2} \mathbb{E}_{t \sim [1,T], \mathbf{x}, \mathbf{z}_t \sim \mathcal{N}(\mathbf{x}, \sigma_t^2 I)} \left[w(\sigma_t) \cdot \left\| \mathbf{s}_\theta(\mathbf{z}_t, \sigma_t) + \frac{\mathbf{z}_t - \mathbf{x}}{\sigma_t^2} \right\|_2^2 \right]. \tag{33}$$

Empirically, it is observed that $\|\mathbf{s}_\theta(\mathbf{x}, \sigma)\|_2 \propto 1/\sigma$, suggesting $w(\sigma) = \sigma^2$ balances the magnitude across scales.

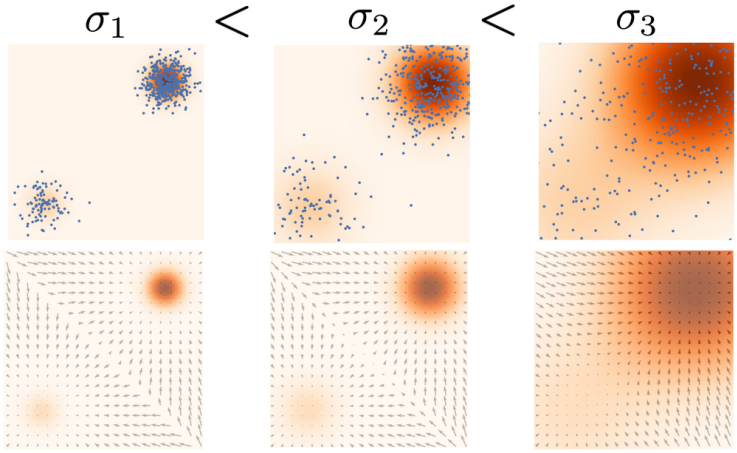


Figure 1.18. Score estimation at multiple noise levels. For each noise scale σ_t, a score network $\mathbf{s}_\theta(\mathbf{x}, \sigma_t)$ is trained to approximate the score $\nabla_{\mathbf{z}_t} \log q(\mathbf{z}_t \mid \mathbf{x})$ of the corresponding noisy distribution. The top row shows the perturbed data distributions for increasing noise levels $\sigma_1 < \sigma_2 < \sigma_3$, and the bottom row shows the estimated scores overlaid on the data density.

This training setup defines the *Noise Conditional Score Network* (NCSN), a model that can estimate the score of perturbed data distributions across multiple noise levels. Figure 1.18 shows a visualization comparing the different noise scales and the estimated score for each scale.

As shown by [Kingma and Gao 2023], the noise prediction loss from Equation 27 is equivalent to the score matching objective when $w(\sigma_t) = \sigma_t^2$.

1.4.4.1. Annealed Langevin dynamics for sampling

Algorithm 3 Annealed Langevin Dynamics (adapted from [Song et al. 2021b])

Input: Noise levels $\{\sigma_t\}_{t=1}^{T}$, step size parameter α, number of steps K
Output: Sample vector $\mathbf{x}$ from the target distribution
Initialize $\tilde{\mathbf{z}}_0 \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ ▷ Initial Gaussian noise
for $t \in \{1, \ldots, T\}$ **do**
    $\varepsilon_t \leftarrow \alpha \cdot \sigma_t^2 / \sigma_T^2$ ▷ Compute adaptive step size
    for $i \in \{1, \ldots, K\}$ **do**
        $\omega_i \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ ▷ Sample Gaussian noise
        $\tilde{\mathbf{z}}_i \leftarrow \tilde{\mathbf{z}}_{i-1} + \frac{\varepsilon_t}{2} \mathbf{s}_\theta(\tilde{\mathbf{z}}_{i-1}, \sigma_t) + \sqrt{\varepsilon_t}\, \omega_i$ ▷ Langevin update step
    end for
    $\tilde{\mathbf{z}}_0 \leftarrow \tilde{\mathbf{z}}_K$ ▷ Prepare for next noise scale
end for
$\mathbf{x} \leftarrow \tilde{\mathbf{z}}_K$
return x

Once trained, the NCSN can generate data samples via *annealed Langevin dynamics*. This method applies Langevin sampling iteratively, starting from a high noise level σ_T and gradually reducing it down to σ_1. The complete procedure is presented in Algorithm 3. This annealing procedure allows the model to explore the space broadly and then refine details at lower noise levels, overcoming the limitations of standard Langevin dynamics in low-density or disconnected regions.

1.5. Denoising neural network architecture

The denoising neural network is the core component of diffusion models. It learns to reverse the corruption process by estimating the noise added to the data at a given noise level. This model is trained using pairs of noisy inputs and corresponding targets, and is used during sampling to progressively reconstruct clean data from noise.

The efficacy of a diffusion model is critically dependent on the capacity of its neural network to accurately predict the noise component ε from a noisy input $\mathbf{z}_t$ at a given timestep t. This network, denoted as $\varepsilon_\theta(\mathbf{z}_t, t)$ or $\mathbf{s}_\theta(\mathbf{z}_t, \sigma_t)$, serves as the parameterized function approximator that learns to reverse the forward diffusion process. While various architectures could theoretically be employed, the de facto standard, established by [Ho et al. 2020] and since refined, is a time-conditional U-Net architecture. This choice is motivated by the U-Net's proven success in image-to-image translation tasks, its inherent multi-scale processing, and its ability to preserve high-fidelity spatial information.

1.5.1. The U-Net backbone

The foundational architecture is the U-Net, originally proposed by [Ronneberger et al. 2015] for biomedical image segmentation. Its design consists of two main paths: a contracting (or encoder) path and an expansive (or decoder) path, forming a characteristic "U" shape (see Figure 1.19).

The contracting path follows a typical convolutional network structure. It consists

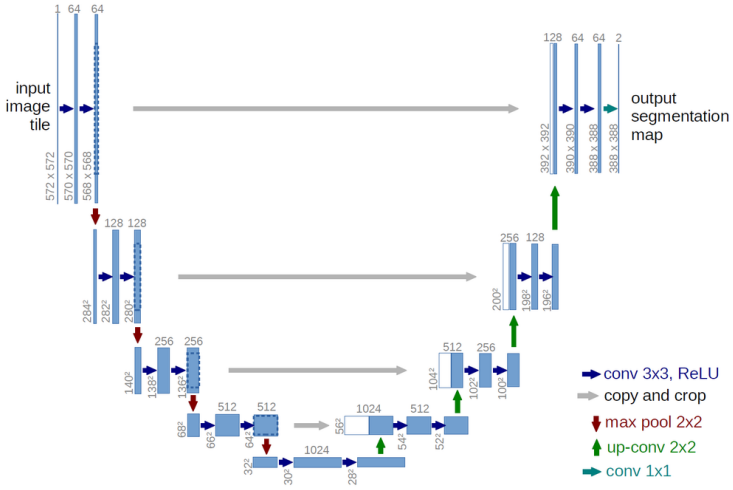


Figure 1.19. The illustration of the original U-Net architecture. The U-shape is clearly visible with a contracting path on the left and an expansive path on the right. Arrows indicate skip connections from the encoder blocks to the decoder blocks. Source: [Ronneberger et al. 2015].

of repeated blocks of standard operations, typically two 3x3 convolutions, each followed by a normalization layer and a non-linear activation function. After each block, a downsampling operation halves the spatial dimensions of the feature maps while doubling the number of feature channels. This process allows the network to capture contextual information at progressively coarser spatial resolutions.

The expansive path is a mirror image of the contracting path. It systematically upsamples the feature maps, typically using a transposed convolution or an upsample-then-convolve operation. A crucial innovation of the U-Net is the use of skip connections, which concatenate the feature maps from the corresponding level in the contracting path with the upsampled feature maps in the expansive path. These connections provide the decoder with high-resolution spatial information from the early layers, which is vital for precise localization and the reconstruction of fine-grained details that would otherwise be lost during downsampling. The final layer maps the feature channels to the desired output size—in this case, the predicted noise map ε_θ.

1.5.2. Architectural enhancements for diffusion

The standard U-Net is adapted and enhanced with several critical components to meet the specific demands of the diffusion process. These enhancements are key to the remarkable performance of modern generative models.

Timestep conditioning. The network's prediction must depend on the timestep t, as the noise to be estimated varies with it. Simply feeding t as an integer is ineffective. Inspired by Transformer positional encodings [Vaswani et al. 2017], a sinusoidal embedding transforms t into a high-dimensional vector, which is processed by a small MLP and added to the feature maps at each residual block. This lets the network modulate its behavior across the hierarchy according to the noise level.

Attention mechanisms. Convolutions are effective for local patterns but have

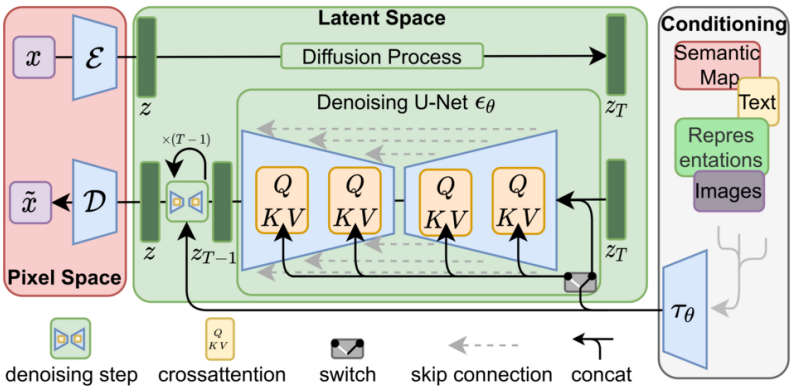


Figure 1.20. Overview of the Stable Diffusion architecture. A denoising UNet operates in latent space, guided by cross-attention over conditioning inputs such as text or image features. The process starts from Gaussian noise and iteratively reconstructs a latent representation, which is decoded into pixel space. Source: [Rombach et al. 2021].

limited receptive fields. To capture long-range dependencies and global context, self-attention blocks are inserted into lower-resolution layers of the U-Net [Dhariwal and Nichol 2021a], where cost is lower and global information is more critical. These blocks enable interactions between distant spatial locations, improving structure, symmetry, and coherence.

Conditional generation via cross-attention. For tasks like text-to-image synthesis, the U-Net is conditioned on external inputs via cross-attention, a key feature of the Latent Diffusion Model (LDM) [Rombach et al. 2021]. Here, the spatial feature maps from the U-Net act as the query vectors, while the conditioning vector (e.g., a text embedding from a pre-trained CLIP model) provides the key and value vectors. Cross-attention layers are distributed across the U-Net, typically alongside self-attention, aligning the denoising process with semantic guidance across scales and effectively steering the generation. Figure 1.20 shows what this conditioning looks like.

1.6. Architectural and methodological advancements

Since their initial formulation, as detailed in Sections 1.3 and 1.4, diffusion models have undergone numerous enhancements aimed at improving sample quality, generation speed, and conditional control. These advancements have transformed them from theoretical curiosities into state-of-the-art generative tools. In this section, we highlight several seminal lines of improvement.

1.6.1. Accelerating sampling via advanced solvers

A primary limitation of early diffusion models like DDPMs was the computationally expensive sampling process, often requiring hundreds or even thousands of sequential network evaluations. Considerable effort has been devoted to accelerating this process without degrading sample quality.

The work of [Song et al. 2021b] introduced a foundational view by framing diffusion as a stochastic differential equation (SDE), showing that DDPMs' discrete forward

and reverse processes are discretizations of a continuous-time SDE. This yields a reverse-time ordinary differential equation (ODE) whose solution maps noise into data, reframing sampling as a numerical ODE-solving problem and enabling the use of advanced solvers.

One of the earliest and most impactful methods derived from this view was the Denoising Diffusion Implicit Model (DDIM) [Song et al. 2021a], which defined a non-Markovian forward process leading to a deterministic reverse path. Unlike the stochastic, step-by-step sampling of DDPMs, DDIM allows for large, deterministic transitions through latent space, drastically reducing the number of steps (e.g., from 1000 to 50) while maintaining competitive image quality.

Expanding on this ODE-based direction, DPM-Solver [Lu et al. 2022] and DPM-Solver++ [Lu et al. 2023] developed high-order, semi-linear solvers tailored to the structure of the diffusion ODE. These solvers analytically handle the linear components and approximate the non-linear score term using high-order polynomials. As a result, they achieve accurate integration with just 10–20 steps—an order-of-magnitude improvement over earlier approaches.

More recently, Rectified Flow [Liu et al. 2023] proposed a conceptual shift: instead of simulating denoising, it learns a velocity field that maps noise to data along nearly straight lines in latent space. This simplification allows even basic Euler solvers to perform well, offering both speed and a fresh generative modeling paradigm.

1.6.2. Enhancing controllability with guidance

Enabling precise user control over the generated output is essential for many applications. A key milestone in this direction was Classifier Guidance [Dhariwal and Nichol 2021b], which introduced a mechanism to steer generation using gradients from an external, pre-trained classifier. At each denoising step, the gradient of the classifier's output with respect to the current noisy sample $\mathbf{z}_t$ is added to the diffusion model's noise estimate, effectively guiding the generation towards a desired class.

While effective, this approach depends on a separate noise-aware classifier. Classifier-Free Guidance (CFG) [Ho and Salimans 2021] offered a more practical and now widely adopted alternative. It eliminates the need for an external classifier by training the diffusion model on both conditional inputs (e.g., text prompts) and a null or "unconditional" context. During inference, the model computes both a conditional noise prediction $\varepsilon_\theta(\mathbf{x}_t, c)$ and an unconditional one $\varepsilon_\theta(\mathbf{x}_t, \emptyset)$, and then interpolates between them. This mechanism significantly boosts controllability and fidelity, becoming a standard component in modern systems like Stable Diffusion.

Recent works, including Extended Classifier-Free Guidance [Chung et al. 2025] and Free Guidance [Tang et al. 2025], have refined CFG by analyzing its trade-offs. These studies noted issues like mode collapse at high guidance scales and proposed sampling adjustments to balance prompt adherence and output diversity. Rather than introducing a new paradigm, their contribution lies in improving CFG's robustness and generalization across use cases.

1.6.3. Efficient high-resolution synthesis in latent space

Scaling diffusion models to high-resolution images is computationally prohibitive in pixel space due to the quadratic scaling of self-attention and memory costs. Latent Diffusion Models (LDMs) [Rombach et al. 2021] provided a landmark contribution by shifting the diffusion process from the high-dimensional pixel space to a compact, learned latent space. Their approach is a two-stage process: first, a powerful autoencoder is trained to compress images into a semantically rich but spatially smaller latent representation. Second, a diffusion model is trained exclusively in this latent space. By operating on these compressed latents, the LDM can generate high-resolution outputs with a fraction of the computational resources required by pixel-space models. This decoupling of perceptual compression and generative modeling was a key enabler for the widespread accessibility of high-quality image synthesis.

1.6.4. Architectural evolution: beyond the U-Net

A seminal contribution from [Peebles and Xie 2023] was the introduction of the Diffusion Transformer (DiT). This work challenged the necessity of the convolutional U-Net backbone. Their key insight was to replace the U-Net entirely with a standard Transformer architecture. The model operates on latent patches of the image, similar to a Vision Transformer [Dosovitskiy et al. 2021]. They demonstrated that a well-designed Transformer, when scaled up in terms of depth, width, and training data, can outperform the best U-Net-based diffusion models on class-conditional image synthesis benchmarks. This was a landmark result, suggesting that a unified Transformer-based architecture could be a scalable and superior foundation for future generative models across modalities.

1.7. Applications of Diffusion Models

Table 1.1. Prominent Techniques and Their Application in Industrial Diffusion Models

Technique	Industrial Solution / Model	Resource
Latent Diffusion	Stable Diffusion (Stability AI, Runway, LMU Munich)	[Rombach et al. 2021]
Classifier-Free Guidance	DALL-E 2 (OpenAI)	[Ramesh et al. 2022]
Cascaded Diffusion & Strong Text Encoders	Imagen (Google Research)	[Saharia et al. 2022]
Diffusion Transformer Architecture	Sora (OpenAI)	OpenAI (2024). *Video generation models as world simulators*.[2]
Fast ODE/SDE Solvers (DDIM, DPM-Solver)	Stable Diffusion Ecosystem (Hugging Face Diffusers)	Diffusers: State-of-the-art diffusion models[3]

Diffusion models have demonstrated remarkable versatility, extending far beyond image synthesis into a wide array of scientific and industrial domains. Their strong generative capabilities, ability to model complex distributions, and compatibility with con-

[2]Technical Report: `https://openai.com/research/video-generation-models-as-world-simulators`

[3]Hugging Face Diffusers Documentation: `https://huggingface.co/docs/diffusers/index`

ditioning mechanisms make them a compelling choice for tasks involving structured or multimodal data. Below, we outline key areas where diffusion models are defining the state of the art. A more complete list of applications in a wide range of areas can be found in [Yang et al. 2024]. To illustrate this direct lineage from academic research to practical application, Table 1.1 summarizes several of these key techniques and identifies their implementation in well-known generative models. Each entry is accompanied by a reference to the official technical report or foundational paper, providing a verifiable link between the concepts and their real-world impact.

1.7.1. Diffusion models in database applications

Diffusion models have become essential tools in database applications, providing controllable synthetic data generation, robust imputation, and hybrid vector-relational capabilities. [Liu et al. 2024] showed that SQL-like predicates can steer the denoising process for workload replay or privacy-preserving data sharing, while FinDiff [Sattarov et al. 2023] demonstrated superior utility-privacy trade-offs compared to GAN baselines in regulated finance applications.

For data repair and augmentation, [Villaizán-Vallelado et al. 2025] combined transformer denoisers with dynamic masking to outperform VAE and GAN methods on supporting both data imputation and synthetic data generation while maintaining privacy. [Li et al. 2025] identified four key research themes: data augmentation, data imputation, trustworthy synthesis, and anomaly detection, while highlighting ongoing challenges in mixed-type encoding, fairness, and scalability.

On the systems side, ACORN [Patel et al. 2024] introduced an in-database index that unifies vector search with relational queries, enabling efficient hybrid workloads over vector data (embedded images, text, and video) as well as structured data, such as attributes and keywords. Regarding privacy concerns, [Wu et al. 2025] addressed privacy vulnerabilities in diffusion models used for tabular data synthesis. Their method demonstrated superior capability in detecting privacy leaks in diffusion-based tabular data synthesis compared to traditional heuristic evaluation methods.

[Shi et al. 2025] addresses the critical need for synthetic tabular data generation in machine learning, where real datasets often suffer from scarcity, privacy concerns, and class imbalance issues. The authors organize methods into three main categories (traditional generation, diffusion models, and LLM-based approaches), provide a complete pipeline overview from generation through evaluation, and identify key challenges and real-world applications.

1.7.2. Synthetic data for tabular domains

In enterprise settings, generating high-fidelity synthetic data is critical for data augmentation, privacy preservation, and simulation. Diffusion models have been adapted to handle heterogeneous tabular data. The contribution of methods like TabDDPM [Kotelnikov et al. 2023] was to design a diffusion process specifically for tabular formats containing a mix of continuous and categorical features. This is achieved by using specialized forward noising schemes for each data type and replacing the convolutional U-Net with a standard Multi-Layer Perceptron (MLP) architecture suited for non-spatial data, thereby enabling

realistic synthetic table generation.

1.7.3. Text-to-image and text-to-video generation

Perhaps the most culturally significant application of diffusion models is in conditional generation from natural language. The key contribution of models like Stable Diffusion [Rombach et al. 2021] was to make high-resolution synthesis computationally tractable by performing the diffusion process in a compressed latent space learned by a powerful autoencoder. Concurrently, models like Imagen [Saharia et al. 2022] demonstrated another critical insight: the power of leveraging large, pre-trained language models as text encoders, combined with a cascaded diffusion architecture that progressively increases image resolution, leading to unprecedented photorealism and semantic fidelity.

This paradigm has been naturally extended to *text-to-video generation*. The primary challenge here is maintaining temporal consistency across frames. Seminal works like Videofusion [Luo et al. 2023] and Video Diffusion Models [Ho et al. 2022] contributed solutions by extending the image generation architecture with spatiotemporal attention mechanisms. Their key innovation was to enable the model to attend not only to spatial information within a frame but also to temporal information across frames, ensuring that object identity and motion remain coherent over time.

1.7.4. Bioinformatics and molecular design

In the life sciences, diffusion models are driving breakthroughs in problems involving complex 3D structures. DiffDock [Corso et al. 2023] reframed molecular docking as a generative modeling task, predicting how a small molecule binds to a protein. It diffuses over ligand poses, including translational, rotational, and torsional degrees of freedom, to efficiently sample realistic binding conformations. In *de novo* protein design, RFdiffusion [Watson et al. 2023] made a landmark contribution by learning to generate novel, functional protein backbones. It applies a diffusion process directly to the 3D coordinates and orientations of amino acid residues, enabling the design of complex protein structures conditioned on functional motifs or structural constraints.

1.7.5. Text and language modeling

While Transformers dominate language modeling, diffusion models offer a compelling alternative paradigm. The key contribution of Diffusion-LM [Li et al. 2022] was to successfully adapt the continuous denoising process to the discrete domain of text. It achieves this by first embedding discrete word tokens into a continuous space, performing the diffusion process on these embeddings, and then rounding the denoised embeddings back to the nearest token in the vocabulary. This non-autoregressive approach offers unique benefits, such as iterative refinement and strong performance on controlled text generation and infilling tasks, where the model must generate text consistent with a given prefix and suffix.

1.7.6. Graph generation

Generative modeling for graphs is essential in areas like social network analysis, drug discovery, and circuit design. The main challenge is jointly modeling a graph's discrete

structure (nodes and edges) and its continuous attributes. DiGress [Vignac et al. 2023] addresses this by proposing a unified framework that diffuses both the adjacency matrix and node/edge features. This joint denoising process captures the interplay between structure and attributes, enabling the generation of realistic and diverse graphs from a simple noise prior.

1.8. Conclusion

In this work, we presented a comprehensive and accessible overview of diffusion models, tracing their development from foundational probabilistic principles to their current role as a leading paradigm in generative artificial intelligence. We began by exploring the theoretical origins of Denoising Diffusion Probabilistic Models (DDPMs) and score-based generative models, showing how both perspectives converge on the goal of learning to reverse a noise process. This unification underpins the core mechanism of diffusion models: training a neural network to denoise samples corrupted through a carefully designed diffusion process.

The practical success of diffusion models is largely attributed to architectural innovations. Central to this is the U-Net backbone, augmented with time-step embeddings and cross-attention modules, enabling both temporal coherence and external conditioning. Alongside this, methodological advances such as Classifier-Free Guidance have improved controllability, while solvers like DDIM and DPM-Solver++ dramatically reduce the sampling time. Latent Diffusion Models (LDMs) further extended the applicability of these techniques to high-resolution outputs with manageable computational cost.

These developments have spurred a broad spectrum of real-world applications. Diffusion models now power systems in domains ranging from image and video generation to protein design, molecular docking, and tabular data synthesis. Their flexibility in modeling complex, multimodal, and structured data highlights their versatility and transformative potential. As the field moves forward, open challenges remain in efficiency, compositionality, and alignment, ensuring that diffusion models will continue to be an active area of research and innovation.

References

[Aali et al. 2023] Aali, A., Arvinte, M., Kumar, S., and Tamir, J. I. (2023). Solving inverse problems with score-based generative priors learned from noisy data. In *2023 57th Asilomar Conference on Signals, Systems, and Computers*, pages 837–843.

[Bishop and Bishop 2024] Bishop, C. M. and Bishop, H. (2024). *Deep Learning - Foundations and Concepts*. Springer.

[Chung et al. 2023] Chung, H., Kim, J., Mccann, M. T., Klasky, M. L., and Ye, J. C. (2023). Diffusion posterior sampling for general noisy inverse problems. In *The Eleventh International Conference on Learning Representations*.

[Chung et al. 2025] Chung, H., Kim, J., Park, G. Y., Nam, H., and Ye, J. C. (2025). CFG++: Manifold-constrained classifier free guidance for diffusion models. In *The Thirteenth International Conference on Learning Representations*.

[Chung et al. 2022] Chung, H., Sim, B., Ryu, D., and Ye, J. C. (2022). Improving diffusion models for inverse problems using manifold constraints. In Oh, A. H., Agarwal, A., Belgrave, D., and Cho, K., editors, *Advances in Neural Information Processing Systems*.

[Corso et al. 2023] Corso, G., Stärk, H., Jing, B., Barzilay, R., and Jaakkola, T. S. (2023). Diffdock: Diffusion steps, twists, and turns for molecular docking. In *The Eleventh International Conference on Learning Representations*.

[Cox and Hinkley 1974] Cox, D. and Hinkley, D. (1974). *Theoretical Statistics*. Chapman and Hall/CRC, New York, 1st edition.

[Dhariwal and Nichol 2021a] Dhariwal, P. and Nichol, A. (2021a). Diffusion models beat gans on image synthesis. *Advances in neural information processing systems*, 34:8780–8794.

[Dhariwal and Nichol 2021b] Dhariwal, P. and Nichol, A. Q. (2021b). Diffusion models beat GANs on image synthesis. In Beygelzimer, A., Dauphin, Y., Liang, P., and Vaughan, J. W., editors, *Advances in Neural Information Processing Systems*.

[Dinh et al. 2015] Dinh, L., Krueger, D., and Bengio, Y. (2015). NICE: non-linear independent components estimation. In Bengio, Y. and LeCun, Y., editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Workshop Track Proceedings*.

[Dosovitskiy et al. 2021] Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., Uszkoreit, J., and Houlsby, N. (2021). An image is worth 16x16 words: Transformers for image recognition at scale. In *International Conference on Learning Representations*.

[Fefferman et al. 2013] Fefferman, C., Mitter, S., and Narayanan, H. (2013). Testing the manifold hypothesis.

[Gong et al. 2022] Gong, S., Li, M., Feng, J., Wu, Z., and Kong, L. (2022). Diffuseq: Sequence to sequence text generation with diffusion models. *arXiv preprint arXiv:2210.08933*.

[Goodfellow et al. 2014] Goodfellow, I. J., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. (2014). Generative adversarial nets. In *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'14, page 2672–2680, Cambridge, MA, USA. MIT Press.

[Ho et al. 2020] Ho, J., Jain, A., and Abbeel, P. (2020). Denoising diffusion probabilistic models. In *Proceedings of the 34th International Conference on Neural Information Processing Systems*, NIPS '20, Red Hook, NY, USA. Curran Associates Inc.

[Ho and Salimans 2021] Ho, J. and Salimans, T. (2021). Classifier-free diffusion guidance. In *NeurIPS 2021 Workshop on Deep Generative Models and Downstream Applications*.

[Ho et al. 2022] Ho, J., Salimans, T., Gritsenko, A. A., Chan, W., Norouzi, M., and Fleet, D. J. (2022). Video diffusion models. In Oh, A. H., Agarwal, A., Belgrave, D., and Cho, K., editors, *Advances in Neural Information Processing Systems*.

[Hyvärinen 2005] Hyvärinen, A. (2005). Estimation of non-normalized statistical models by score matching. *J. Mach. Learn. Res.*, 6:695–709.

[Jordan et al. 1998] Jordan, M. I., Ghahramani, Z., Jaakkola, T. S., and Saul, L. K. (1998). *An Introduction to Variational Methods for Graphical Models*, pages 105–161. Springer Netherlands, Dordrecht.

[Kingma and Gao 2023] Kingma, D. P. and Gao, R. (2023). Understanding diffusion objectives as the ELBO with simple data augmentation. In *Thirty-seventh Conference on Neural Information Processing Systems*.

[Kingma et al. 2021] Kingma, D. P., Salimans, T., Poole, B., and Ho, J. (2021). On density estimation with diffusion models. In Beygelzimer, A., Dauphin, Y., Liang, P., and Vaughan, J. W., editors, *Advances in Neural Information Processing Systems*.

[Kingma and Welling 2022] Kingma, D. P. and Welling, M. (2022). Auto-encoding variational bayes.

[Kotelnikov et al. 2023] Kotelnikov, A., Baranchuk, D., Rubachev, I., and Babenko, A. (2023). TabDDPM: Modelling tabular data with diffusion models.

[Li et al. 2022] Li, X. L., Thickstun, J., Gulrajani, I., Liang, P., and Hashimoto, T. (2022). Diffusion-LM improves controllable text generation. In Oh, A. H., Agarwal, A., Belgrave, D., and Cho, K., editors, *Advances in Neural Information Processing Systems*.

[Li et al. 2025] Li, Z., Huang, Q., Yang, L., Shi, J., Yang, Z., van Stein, N., Bäck, T., and van Leeuwen, M. (2025). Diffusion models for tabular data: Challenges, current progress, and future directions.

[Liu et al. 2024] Liu, T., Fan, J., Tang, N., Li, G., and Du, X. (2024). Controllable tabular data synthesis using diffusion models. *Proc. ACM Manag. Data*, 2(1).

[Liu et al. 2023] Liu, X., Gong, C., and qiang liu (2023). Flow straight and fast: Learning to generate and transfer data with rectified flow. In *The Eleventh International Conference on Learning Representations*.

[Lu et al. 2022] Lu, C., Zhou, Y., Bao, F., Chen, J., Li, C., and Zhu, J. (2022). DPM-solver: A fast ODE solver for diffusion probabilistic model sampling in around 10 steps. In Oh, A. H., Agarwal, A., Belgrave, D., and Cho, K., editors, *Advances in Neural Information Processing Systems*.

[Lu et al. 2023] Lu, C., Zhou, Y., Bao, F., Chen, J., Li, C., and Zhu, J. (2023). DPM-solver++: Fast solver for guided sampling of diffusion probabilistic models.

[Luo et al. 2023] Luo, Z., Chen, D., Zhang, Y., Huang, Y., Wang, L., Shen, Y., Zhao, D., Zhou, J., and Tan, T. (2023). Videofusion: Decomposed diffusion models for high-quality video generation. *arXiv preprint arXiv:2303.08320*.

[Murphy 2022] Murphy, K. P. (2022). *Probabilistic Machine Learning: An introduction*. MIT Press.

[Nichol and Dhariwal 2021] Nichol, A. Q. and Dhariwal, P. (2021). Improved denoising diffusion probabilistic models.

[Patel et al. 2024] Patel, L., Kraft, P., Guestrin, C., and Zaharia, M. (2024). Acorn: Performant and predicate-agnostic search over vector embeddings and structured data. *Proc. ACM Manag. Data*, 2(3).

[Peebles and Xie 2023] Peebles, W. and Xie, S. (2023). Scalable diffusion models with transformers. In *2023 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 4172–4182.

[Prince 2023] Prince, S. J. (2023). *Understanding Deep Learning*. The MIT Press.

[Ramesh et al. 2022] Ramesh, A., Dhariwal, P., Nichol, A., Chu, C., and Chen, M. (2022). Hierarchical text-conditional image generation with clip latents.

[Rombach et al. 2021] Rombach, R., Blattmann, A., Lorenz, D., Esser, P., and Ommer, B. (2021). High-resolution image synthesis with latent diffusion models.

[Ronneberger et al. 2015] Ronneberger, O., Fischer, P., and Brox, T. (2015). U-net: Convolutional networks for biomedical image segmentation. In Navab, N., Hornegger, J., Wells, W. M., and Frangi, A. F., editors, *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015*, pages 234–241, Cham. Springer International Publishing.

[Saharia et al. 2022] Saharia, C., Chan, W., Saxena, S., Lit, L., Whang, J., Denton, E., Ghasemipour, S. K. S., Ayan, B. K., Mahdavi, S. S., Gontijo-Lopes, R., Salimans, T., Ho, J., Fleet, D. J., and Norouzi, M. (2022). Photorealistic text-to-image diffusion models with deep language understanding. In *Proceedings of the 36th International Conference on Neural Information Processing Systems*, NIPS '22, Red Hook, NY, USA. Curran Associates Inc.

[Sattarov et al. 2023] Sattarov, T., Schreyer, M., and Borth, D. (2023). Findiff: Diffusion models for financial tabular data generation. In *Proceedings of the Fourth ACM International Conference on AI in Finance*, ICAIF '23, page 64–72, New York, NY, USA. Association for Computing Machinery.

[Shi et al. 2025] Shi, R., Wang, Y., Du, M., Shen, X., Chang, Y., and Wang, X. (2025). A comprehensive survey of synthetic tabular data generation.

[Sohl-Dickstein et al. 2015] Sohl-Dickstein, J., Weiss, E., Maheswaranathan, N., and Ganguli, S. (2015). Deep unsupervised learning using nonequilibrium thermodynamics. In *International conference on machine learning*, pages 2256–2265. pmlr.

[Song et al. 2021a] Song, J., Meng, C., and Ermon, S. (2021a). Denoising diffusion implicit models. In *International Conference on Learning Representations*.

[Song and Ermon 2019] Song, Y. and Ermon, S. (2019). *Generative modeling by estimating gradients of the data distribution*. Curran Associates Inc., Red Hook, NY, USA.

[Song et al. 2019] Song, Y., Garg, S., Shi, J., and Ermon, S. (2019). Sliced score matching: A scalable approach to density and score estimation. *CoRR*, abs/1905.07088.

[Song et al. 2021b] Song, Y., Sohl-Dickstein, J., Kingma, D. P., Kumar, A., Ermon, S., and Poole, B. (2021b). Score-based generative modeling through stochastic differential equations. In *International Conference on Learning Representations*.

[Tang et al. 2025] Tang, Z., Bao, J., Chen, D., and Guo, B. (2025). Diffusion models without classifier-free guidance.

[Vaswani et al. 2017] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. (2017). Attention is all you need. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, page 6000–6010, Red Hook, NY, USA. Curran Associates Inc.

[Vignac et al. 2023] Vignac, C., Krawczuk, I., Siraudin, A., Wang, B., Cevher, V., and Frossard, P. (2023). Digress: Discrete denoising diffusion for graph generation. In *The Eleventh International Conference on Learning Representations*.

[Villaizán-Vallelado et al. 2025] Villaizán-Vallelado, M., Salvatori, M., Segura, C., and Arapakis, I. (2025). Diffusion models for tabular data imputation and synthetic data generation. *ACM Trans. Knowl. Discov. Data*, 19(6).

[Vincent 2011] Vincent, P. (2011). A connection between score matching and denoising autoencoders. *Neural Comput.*, 23(7):1661–1674.

[Wainwright and Jordan 2008] Wainwright, M. J. and Jordan, M. I. (2008). Graphical models, exponential families, and variational inference. *Found. Trends Mach. Learn.*, 1(1–2):1–305.

[Watson et al. 2023] Watson, J. L., Juergens, D., Bennett, N. R., Trippe, B. L., Yim, J., Eisenach, H. E., Ahern, W., Borst, A. J., Ragotte, R. J., Milles, L. F., Wicky, B. I. M., Hanikel, N., Pellock, S. J., Courbet, A., Sheffler, W., Wang, J., Venkatesh, P., Sappington, I., Torres, S. V., Lauko, A., De Bortoli, V., Mathieu, E., Ovchinnikov, S., Barzilay, R., Jaakkola, T. S., DiMaio, F., Baek, M., and Baker, D. (2023). De novo design of protein structure and function with RFdiffusion. *Nature*, 620(7976):1089–1100.

[Wu et al. 2025] Wu, X., Pang, Y., Liu, T., and Wu, S. (2025). Winning the midst challenge: New membership inference attacks on diffusion models for tabular data synthesis.

[Yang et al. 2024] Yang, L., Zhang, Z., Song, Y., Hong, S., Xu, R., Zhao, Y., Zhang, W., Cui, B., and Yang, M.-H. (2024). Diffusion models: A comprehensive survey of methods and applications.

Chapter 2

Introduction to LLM-Based Agents

Eduardo Bezerra

Abstract

This chapter provides an introductory yet comprehensive exploration of large language model (LLM)-based agents, tracing their evolution from early statistical language models to modern, tool-using systems capable of reasoning and acting in complex environments. We review the key architectural advances that enabled emergent capabilities, introduce prompting techniques and interaction patterns such as ReAct, Plan-and-Act, and Pre-Act, and explain how these patterns coordinate multi-step reasoning with external tools. Core mechanisms including tool calling, Retrieval-Augmented Generation (RAG), and Text-to-SQL pipelines are examined. To bridge theory and practice, the chapter is accompanied by a suite of Jupyter notebooks that demonstrate complete end-to-end workflows across the topics presented. The aim is to equip the reader with both the conceptual understanding and hands-on skills necessary to design, implement, and critically assess LLM-based agents in real-world applications.

> Agents represent an exciting and promising new approach to building a wide range of software applications. Agents are autonomous problem-solving entities that are able to flexibly solve problems in complex, dynamic environments, without receiving permanent guidance from the user.
>
> *Jennings & Wooldridge (1998)*

2.1. Introduction

According to the AIMA Book [Russell and Norvig 2021], an *intelligent agent* is an agent that acts rationally. That is, it selects actions that are expected to maximize its performance measure, based on the percept sequence and its built-in knowledge. The quest to such agents has been a central theme in AI since its inception. Early efforts, such as the Logic Theorist [Newell et al. 1956] and Shakey the Robot [Nilsson 1984], focused on *symbolic reasoning* and planning in structured environments. These systems embodied the classical view of agents as entities that perceive, reason, and act to achieve goals. With the advent of the large language model (LLM) era, interest in AI agents has experienced a strong resurgence. Unlike earlier symbolic systems, modern agents are increasingly built using *connectionist paradigms*, in which reasoning is powered by LLMs instead of symbolic logic.

The goal of this chapter is to provide a comprehensive yet accessible introduction to LLM-based agents. We aim to equip the reader with a conceptual understanding of their architectures and interaction strategies, and to demonstrate how they can be used to perform structured tasks over databases and other tools. The chapter is accompanied with hands-on examples in the form of Jupyter notebooks.

This rest of this chapter is organized as follows. Section 2.2 reviews the evolution from statistical language models to modern neural-based LLMs, highlighting the scaling advances that enabled emergent capabilities. Section 2.3 transitions from standalone LLMs to agentic systems, defining LLM-based agents, clarifying "structured tasks," and motivating tool-using architectures. Section 2.4 introduces prompting techniques and interaction patterns that structure multi-step reasoning and tool use. Section 2.5 details the tool-calling mechanism, including tool registration, structured message exchange, and orchestration between the LLM and external systems. Section 2.6 presents Retrieval-Augmented Generation (RAG) as a strategy to ground outputs in authoritative external data. Section 2.7 describes the complete Text-to-SQL pipeline, covering intent parsing, schema linking, value grounding, SQL generation, execution and correction, and answer presentation with interactive refinement. Section 2.8 presents a set of Jupyter notebooks that illustrate the full range of concepts and techniques discussed in the chapter, enabling hands-on experimentation beyond individual components. Finally, Section 2.9 offers concluding observations, discusses limitations and ethical considerations, and highlights directions for further study.

2.2. From Statistical Language Models to Neural-based LLMs

By definition, a *language model* is a system trained to understand and generate human language by learning patterns from large amounts of text data[1]. It predicts what *tokens*, which are small units of text (a whole word, part of a word, or even punctuation), are most likely to come next in a sequence. Then the predicted token can be appended to the sequence, and another token can be predicted. By repeatedly

[1]Although the term "language model" has recently been extended to other modalities such as images, audio, and video, this chapter focuses exclusively on agents that interact with language models for text.

performing this prediction, the model can generate entire sentences, paragraphs, or even full documents. For this reason, a language model is a type of *generative model*.

Early approaches to build language models were mainly probabilistic in nature [Rosenfeld 2000]. These models learns conditional probabilities of co-occurrence of items by counting the frequency of token sequences in a large body of text, called a *corpus*. It was the era of the n-gram models. For example, a bi-gram model (an n-gram with $n = 2$) estimates the probability of the next token (w_i) based only on the previous token (w_{i-1}), i.e., $\Pr(w_i|w_{i-1})$. A tri-gram model ($n = 3$) considers the two preceding tokens, i.e., $\Pr(w_i|w_{i-2}, w_{i-1})$. In general, an n-gram model learns these probabilities by counting the frequency of token sequences in a large body of text, called a *corpus*. For a bi-gram model, the probability would be calculated as:

$$\Pr(w_i|w_{i-1}) = \frac{\text{count}(w_{i-1}, w_i)}{\text{count}(w_{i-1})}$$

In a statistical language model, its *context window* refers to the portion of preceding text the model can consider when predicting the next token. In this case, the context window is fixed to the last $n-1$ tokens. For example, a trigram ($n = 3$) model predicts the next token using only the previous two tokens as context. Tokens outside this window have no influence on the prediction

The probabilistic approach allows the model to predict the most likely next token in a sequence. For instance, after seeing the sequence "the cat", a trigram model might assign a high probability to tokens like "sat" or "is", and low probability to tokens like *barks*. These probability values are computed based on how often those pairs appeared in the training corpus. Earlier probabilistic approaches, such as n-gram models, represented language through discrete frequency counts and conditional probabilities. While effective in certain domains, they were limited by sparsity issues and short context windows. With the advent of the Deep Learning era [Bezerra 2016, LeCun et al. 2015], neural network-based approaches for building language models began to appear in the literature [Bengio et al. 2003, Mikolov et al. 2010, Vaswani et al. 2017]. These new approaches eventually gave rise to what is current known as *Large Language Models* (LLMs).

LLMs are language models with a very large number of parameters, trained using self-supervised learning on vast amounts of text, and capable of leveraging much longer context windows. It is difficult to pinpoint exactly when the adjective "large" began to be consistently attached to the term "language model". Although the expression appears as early as 2018 in reference to models such as BERT [Devlin et al. 2019], it rose to prominence and became the default terminology as model scale increased dramatically, particularly following the release of GPT-3 in 2020. However, the meaning of "large" in LLMs continues to evolve, as each new generation of models surpasses the previous in size and capability.

While the concept of context window is common to both pre-neural and modern neural approaches, its interpretation differs across generations of models.

In statistical language models, each prediction step simply uses the most recent $n-1$ tokens available. In modern neural LLMs, the context window is much larger (thousands or even hundreds of thousands of tokens). Here, the window is a shared "token budget" that includes both the input prompt and the model's generated output. For example, with a 4,096-token context window, if the prompt uses 1,000 tokens, the model can generate up to 3,096 tokens before earlier tokens start to be dropped from consideration (a phenomenon sometimes called *contextual drift*). In both cases, the size of the context window limits how much prior information the model can use at once, directly affecting its ability to maintain coherence, recall details, and follow multi-step instructions.

In modern LLM-based applications, the term *prompt* refers to the input provided to the model at inference time to elicit a desired output. The prompt can contain instructions, examples, constraints, or context information, and it is processed along with the model's learned parameters to determine the generated response. The design of prompts (i.e., what information to include, in what order, and with what phrasing) has a direct impact on the model's behavior and output quality.

A related concept, particularly relevant in multi-turn or agent-based setups, is the *system prompt* (sometimes called a "system message"). This is an instruction or set of instructions, often provided at the start of a conversation, that defines the model's overall role, objectives, tone, and operational constraints. Unlike user prompts that change with each interaction, the system prompt persists as part of the context across turns, acting as a stable guide for the model's responses. In frameworks for build LLM-based applications, the system prompt is explicitly separated from user messages; in these frameworks, it often embeds information about available tools, policies, or domain-specific knowledge.

By the time we discuss prompting techniques in Section 2.4 and tool registration in Section 2.5, both concepts, prompt and system prompt, will serve as foundational elements. They are key to understanding how natural language is transformed into structured model behavior, whether in a simple question-answer scenario or in a complex, multi-step agent workflow.

The context window sets a hard limit on how much information an LLM can use at once. However, scaling up model architecture, data, and compute has empirically revealed a complementary phenomenon, one in which entirely new capabilities emerge. An important observation in the study of LLMs is the emergence of qualitatively new capabilities as model scale increases. This idea echoes Philip W. Anderson's 1972 essay *More Is Different* [Anderson 1972], which argues that increasing the scale and complexity of a system can give rise to qualitatively new phenomena, irreducible to the behavior of its smaller components. In the natural sciences, for example, the principles of chemistry emerge from, but are not reducible to, the laws of quantum physics, and biology in turn exhibits properties not explainable by chemistry alone. Recent studies of large language models reveal similar patterns: as the number of parameters, the amount of training data, and the compute budget increase, new abilities often appear abruptly rather than gradually [Wei et al. 2022a,

Kayhan et al. 2023]. Examples include in-context learning, multi-step reasoning, and code generation. These emergent capabilities mirror Anderson's insight that "more" can indeed be fundamentally "different".

The rapid appearance of these emergent capabilities has fueled an intense period of innovation in model architectures and training strategies. Each successive generation of LLMs not only increased in size but also integrated new techniques, such as improved pretraining objectives, instruction tuning, and reinforcement learning from human feedback, that amplified their reasoning abilities and practical usefulness. This acceleration is evident when looking at the key milestones in LLM development. Figure 2.1 presents a timeline that summarizes the chronological evolution of large language models (LLMs) and LLM-based agents from 2018 to 2025. On the left-hand side of the timeline, we highlight key LLM milestones, beginning with GPT-1 and GPT-2 developed by OpenAI, followed by T5 and PaLM, and later models such as LLaMA, Claude, Gemini, Qwen, DeepSeek R1, and GPT-5. These models reflect the rapid progression in model scale, training data, and emergent properties.

The advances in architecture, scale, and training that led to neural-based LLMs not only improved language modeling accuracy but also unlocked capabilities that go beyond text generation. These capabilities have made it possible to design systems where the LLM is one component in a larger framework, **an agent**, capable of reasoning, planning, and interacting with external tools and environments.

2.3. From LLMs to LLM-based Agents

Although autonomous agents have a long history in AI (see Section 2.1), their resurgence in the LLM era began in late 2022 with the release of ChatGPT, and accelerated in early 2023 with frameworks like LangChain and open-source projects such as AutoGPT and BabyAGI, which showcased how LLMs could plan, use tools, and act autonomously. We define a *LLM-based agent* as a system that is built around a Large Language Model. LLM-based agents are also known as *generative agents*. By being built around a LLM, we mean that the LLM can be considered the "brain" of the agent. This analogy is appropriate for several reasons. First, the LLM processes information from the agent's environment and the user. Second, the LLM uses its understanding of the current context to decide on a course of action. Lastly, the LLM creates natural language outputs, whether to interact with a user or to command other tools.

In the context of AI agents, a *structured task* is a problem or goal that can be broken down into a series of well-defined, discrete steps with clear inputs and outputs. These tasks are typically predictable and follow a specific, predetermined workflow. A LLM-based agent extends its underlying LLM with features such as structured decision-making processes and access to external tools. These capabilities transform the underlying LLM (which is a passive text generator) into a active agent that can perceive, reason, and act in the real world, making them suitable for complex, multi-step structured tasks that require both intelligence and execution [Sapkota et al. 2025].

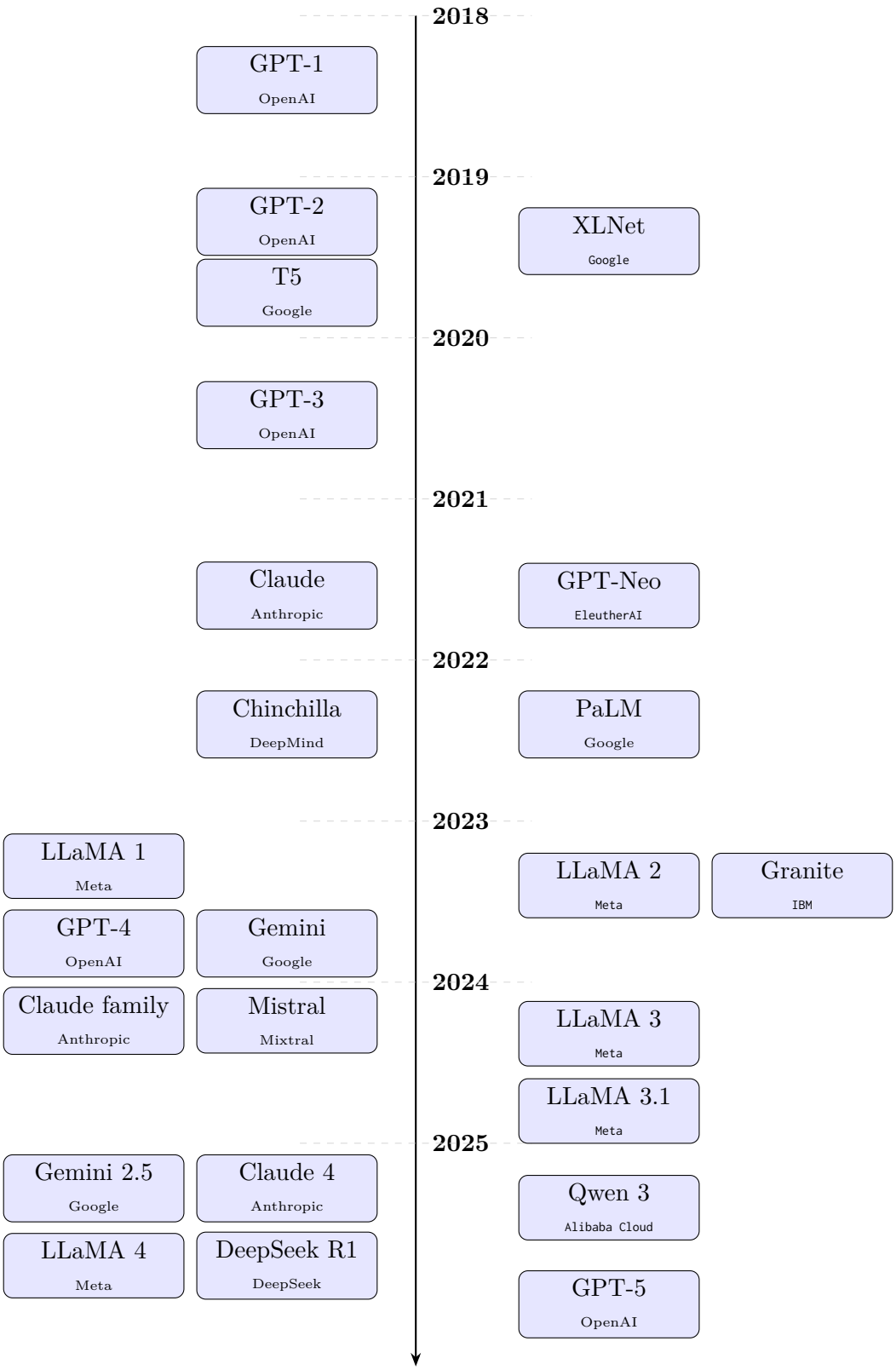


Figure 2.1: Timeline showing the evolution of selected Large Language Models from 2018 to 2025.

2.4. From Prompting Techniques to Interaction Patterns

The idea of guiding a language model's behavior through its input existed in earlier NLP systems, such as sequence-to-sequence models for translation and summarization, but it was often framed in terms of task-specific inputs rather than "prompting". With the release of GPT-2 in 2019, and especially GPT-3 in 2020, the effectiveness of well-crafted natural language prompts in steering large, general-purpose models became widely recognized, sparking a wave of research and practice in *prompting techniques*. Table 2.1 summarizes some notable prompting techniques.

Technique	Description
Zero-shot	Ask the model to perform a task without giving any examples.
Few-shot	Provide a small set of examples to guide output format and style.
Chain-of-thought	Instruct the model to reason step-by-step before answering.
Role	Assign a role/persona to influence tone and perspective.
Self-consistency	Sample multiple reasoning paths and choose the most common answer.
Style/format	Specify required style, tone, or format in the prompt.

Table 2.1: Common prompting techniques for a single LLM invocation.

Most of the prompting techniques listed in Table 2.1 are self-explanatory. One notable exception is *Chain-of-Thought* (CoT), which was the first prompting technique to be formalized in a high-profile paper [Wei et al. 2022b]. CoT refers to phrasing a single prompt in a way that elicits step-by-step reasoning from the model. CoT prompting can be explicit (instructing the model to reason step-by-step) or implicit (designing a question that naturally triggers reasoning). Modern instruction-tuned LLMs often display CoT-like reasoning without the explicit phrase, but the explicit form remains important when one needs to reliably control output style or reasoning depth. Explicit CoT is typically achieved by adding an instruction such as "Let's think step by step" to the prompt, or by supplying intermediate reasoning examples. An illustration of explicit CoT in use is provided in Figure 2.2; note, however, that if only the question "What is the day before the day after tomorrow?" is provided to a modern LLM (without the "Let's think step by step" part), it will most likely generate the reasoning trace along with the final answer.[2]

While simple prompting techniques (Table 2.1) can be effective for direct questions or single-step reasoning, they are often insufficient for tasks that require

[2]This behavior is largely due to the fine-tuning process used in modern instruction-tuned LLMs, which often includes examples of step-by-step reasoning in domains such as mathematics, logic puzzles, and riddles. During reinforcement learning from human feedback (RLHF), annotators tend to reward answers that include clear intermediate reasoning, making such outputs more likely even when no explicit "Let's think step by step" instruction is given.

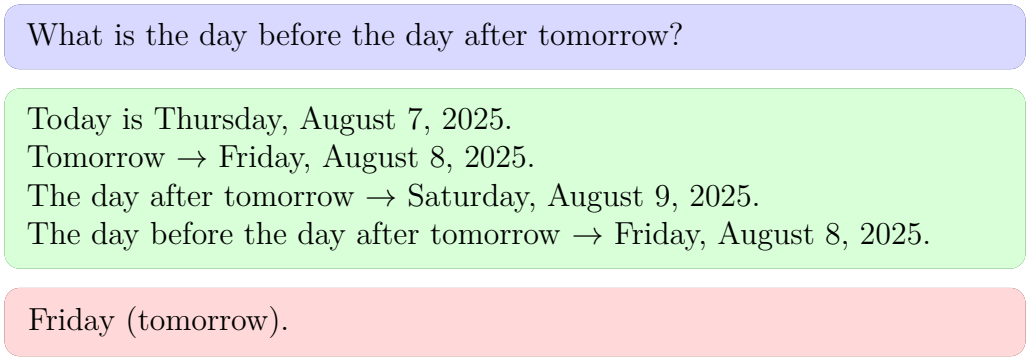


Figure 2.2: Example of explicit Chain-of-Thought reasoning applied to a simple logic puzzle.

multiple reasoning steps, integration of external information, or stateful decision-making. In such cases, the interaction between the model and its environment must follow a structured process rather than a single prompt-response exchange.

An *interaction pattern* is a structured template that defines how a language model agent engages in multi-step interactions (encompassing reasoning, tool use, and memory updates) to perform complex tasks. By specifying the order and format of the model's outputs, such patterns promote consistent behavior and facilitate integration into larger systems. Modern LLM-based agents employ these patterns to coordinate reasoning with external tools (e.g., search engines, database interfaces, and APIs), interpret results, adapt their behavior, and decide whether to invoke additional tools or finalize their responses. The emergence of these interaction patterns has been a key catalyst for the renewed interest in building AI agents in the LLM era.

Examples of interaction patterns include *ReAct*, *Plan-and-Act*, *Pre-Act*, and *Tree-of-Thought*. Table 2.2 summarizes a selection of prominent interaction patterns for LLM-based agents, ordered chronologically by their initial publication date. This timeline helps illustrate how research has progressively explored different strategies for structuring model-environment interaction, from the introduction of *ReAct* in late 2022 to more recent approaches such as *Plan-and-Act* and *Pre-Act* in 2025.

Month/Year	Interaction Pattern	Reference
October/2022	ReAct	[Yao et al. 2022]
May/2023	Tree-of-Thought	[Yao et al. 2023]
March/2025	Plan-and-Act	[Erdogan et al. 2025]
May/2025	Pre-Act	[Hu et al. 2025]

Table 2.2: Timeline of selected interaction patterns with references.

The ReAct (Reason + Act) pattern is designed for scenarios in which large language models (LLMs) must interleave reasoning steps with the use of external

tools to accomplish complex, goal-directed tasks [Yao et al. 2022]. The "Reason" component refers to the model's internal reasoning process, similar to the Chain-of-Thought (CoT) approach. The "Act" component corresponds to moments when the model decides to invoke an external tool (such as a search engine, calculator, or database query) to obtain intermediate results that support the final answer. Because LLMs lack direct perception of their environment, they must be explicitly informed (by the agent) about which tools are available. Once equipped with this knowledge, the model can determine, for each prompt, which tools are required and how to integrate their outputs into its reasoning process.

In general, the reasoning process of a model that uses ReAct consists of three stages: Thought, Action, and Observation.

- *Thought*: In this stage, the model generates an internal reasoning step, similar to CoT, to plan its next move. This step may be triggered directly by the original prompt or by the observation of results produced by a previously invoked tool.
- *Action*: Based on the plan, the model may decide it is necessary to interact with an external tool (such as a search engine, calculator, or code interpreter). In this case, the LLM generates a structured instruction indicating which tool should be called and with what parameters.
- *Observation*: The model receives the tool's output and incorporates it into its subsequent reasoning.

Figure 2.3 illustrates the ReAct pattern through an example in which an LLM-based agent handles a business query about payroll expenses. The sequence of steps can be read from top to bottom, following the colored boxes and their legends. The **Query Box** contains a natural language question about the Sales department's payroll for "last month", illustrating how such queries often omit technical specifics. The two **Reasoning Boxes** depict the agent's internal thought process. In the first reasoning step, the agent interprets "last month" relative to the current date `(August 8, 2025)`, concluding that it refers to `July 2025 (2025-07)`. The **Action Box** shows the agent's reasoning (Call Sales API with normalized parameters) and the textual specification of the tool to be invoked. This specification is then used by the agent to make the actual API call. The **Observation Box** contains the raw API response in JSON format, representing the structured data returned by the external system. In the second reasoning step, the agent analyzes this response and prepares it for presentation. Finally, the Answer Box provides the human-readable output (generated by the LLM), synthesizing all the information into a concise, clear answer.

ReAct operationalized the concept of an "LLM-based agent" by giving LLMs a structured way to think, act, and adapt in real time, turning abstract reasoning into practical, tool-augmented workflows. More recent patterns, such as Plan-and-Act separate planning and execution into distinct LLM modules for improved performance on long-horizon tasks [Erdogan et al. 2025]. Pre-Act further enhances this by continually refining multi-step execution plans during task execution [Rawat et al. 2025].

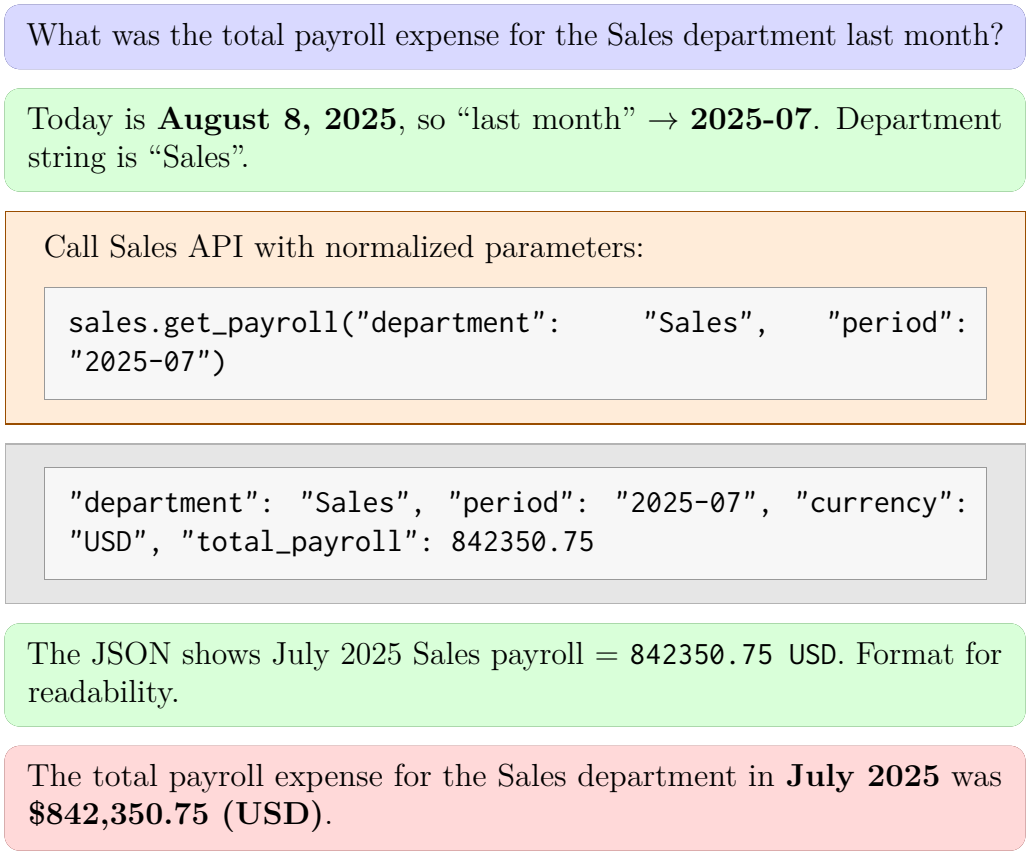


Figure 2.3: ReAct interaction pattern example showing reasoning, action, observation, and answer for a fictional Sales API query.

User Reasoning Action Observation Final Answer

A detailed discussion of these and other emerging patterns is beyond the scope of this chapter; readers are encouraged to consult the corresponding papers for in-depth explanations and examples.

Memory Management in LLM-Based Agents

In the context of LLM-based agents, *memory* refers to mechanisms that allow the agent to retain and reuse information across interactions. While the underlying LLM has no persistent state beyond its current context window, the agent architecture can maintain external memory to simulate continuity over time. Two common forms are:

- **Short-term memory**, typically implemented as a rolling conversation history or working buffer, storing recent messages, tool outputs, and intermediate reasoning steps. This supports coherence within multi-turn tasks and is often truncated or summarized to stay within context window limits.
- **Long-term memory**, implemented via external storage (e.g., databases, vector stores), enabling the agent to recall facts, user preferences, or prior task results across sessions. This is particularly important in personalized assistants and domain-specific agents.

Memory management involves deciding *what* to store, *how* to represent it (raw text, embeddings, structured records), and *when* to retrieve or summarize it. Poorly managed memory can lead to context overflow, forgotten constraints, or irrelevant retrievals. In interaction patterns such as ReAct or Plan-and-Act, effective memory use supports informed decision-making by grounding reasoning steps in prior context.

Although not the focus of this chapter, memory is a critical enabler for sustained, context-aware behavior in LLM-based agents and directly complements the tool calling, RAG, and prompting strategies discussed later in the chapter.

2.5. Tool Calling

In the agent paradigm, a *tool* is any external capability that an LLM can invoke to perform reasoning steps or access information beyond its pretraining. Examples include APIs, search engines, calculators, and SQL databases. *Tool calling* refers to the mechanism by which an LLM interacts with such external functions, APIs, or services by generating *structured messages* with the appropriate parameters. As discussed in Section 2.4, this process often relies on an interaction pattern, such a ReAct.

Before an LLM-based agent can decide that a tool should be invoked, it must be made aware of its existence, capabilities, and usage parameters. This process is known as *tool registration*. This is typically achieved by providing the model with a structured description of each tool (often in JSON or another machine-readable schema) at the start of the interaction or during an initialization step. The specification usually includes the tool's name, purpose, input parameters (with types and constraints), and expected output format. These descriptions are embedded into the model's system prompt or provided via an API that supports dynamic tool registration. By incorporating this metadata into its context, the LLM can reason about which tool is relevant to the current task, how to construct valid calls, and how to interpret the returned results. Examples of tool registration for a Search API and an SQL executor are shown in Listings 2.1 and 2.2, respectively.

```
{
  "name": "search_knowledge_base",
  "description": "Find documents in the domain-specific KB",
  "parameters": {
    "query": { "type": "string", "description": "Search terms" },
    "top_k": { "type": "integer", "description": "Max results", "default": 5 }
  }
}
```

Listing 2.1: Example of tool registration with search API details.

Tool registration itself is an instance of a *structured message exchange*. The agent sends the LLM a machine-readable description of each tool, following a fixed schema (often JSON) that encodes metadata such as the tool's name, purpose, input parameters, and output format.

```
{
  "name": "execute_sql",
  "description": "Run SQL queries on the market_data database",
  "parameters": {
    "sql": { "type": "string", "description": "Valid SQL query" }
  },
  "schema": {
    "tables": {
      "companies": ["company_id", "name", "sector", "market_cap"],
      "stock_prices": ["company_id", "trade_date", "close_price"]
    },
    "relationships": [
      "companies.company_id = stock_prices.company_id"
    ]
  }
}
```

Listing 2.2: Example of tool registration with database schema details.

In general, a *structured message* is a machine-readable data object used for communication either within the agent (for example, between the LLM and its orchestration layer) or between the agent and external systems. It follows a predefined, consistent format (typically JSON or similar) with explicit fields for metadata, parameters, and content, enabling reliable parsing, execution of tool calls, and integration of results across multi-step interactions. Listings 2.1 and 2.2 are examples of structured messages that an agent can send to its underlying LLM.

Figure 2.4 presents a schematic view of the tool calling mechanism. The diagram shows a nested architecture in which the agent (depicted as the outer green container) manages the entire process. Within the agent, the LLM serves as the reasoning component. External tools (e.g., search API, SQL executor, code interpreter) are separate systems that the agent can invoke.

In step 1, a human user, a system task, or even another agent provides the agent with an original prompt, for example: "*Identify the top 10 renewable energy companies by market capitalization in 2025, retrieve their average stock price over the past week, and present the results in a table*". Upon receiving this prompt, the agent processes it, informing the LLM of the relevant tools, policies, and context. This information is passed as a structured message. The LLM interprets the message and decides on the next steps, which may include calling a tool.

In the example from Figure 2.4, the LLM first decides to call a search engine. It returns to the agent the specification for invoking this tool, including the textual query: "*Top 10 renewable energy companies by market capitalization in 2025*" (see Listing 2.4a). The agent executes the search (step 2a), receives the results (step 2b), and appends them to a new structured message for the LLM. Reasoning over this updated context, the LLM determines that it needs information from another tool, this time, an SQL engine. The agent again receives the tool call specification, invokes

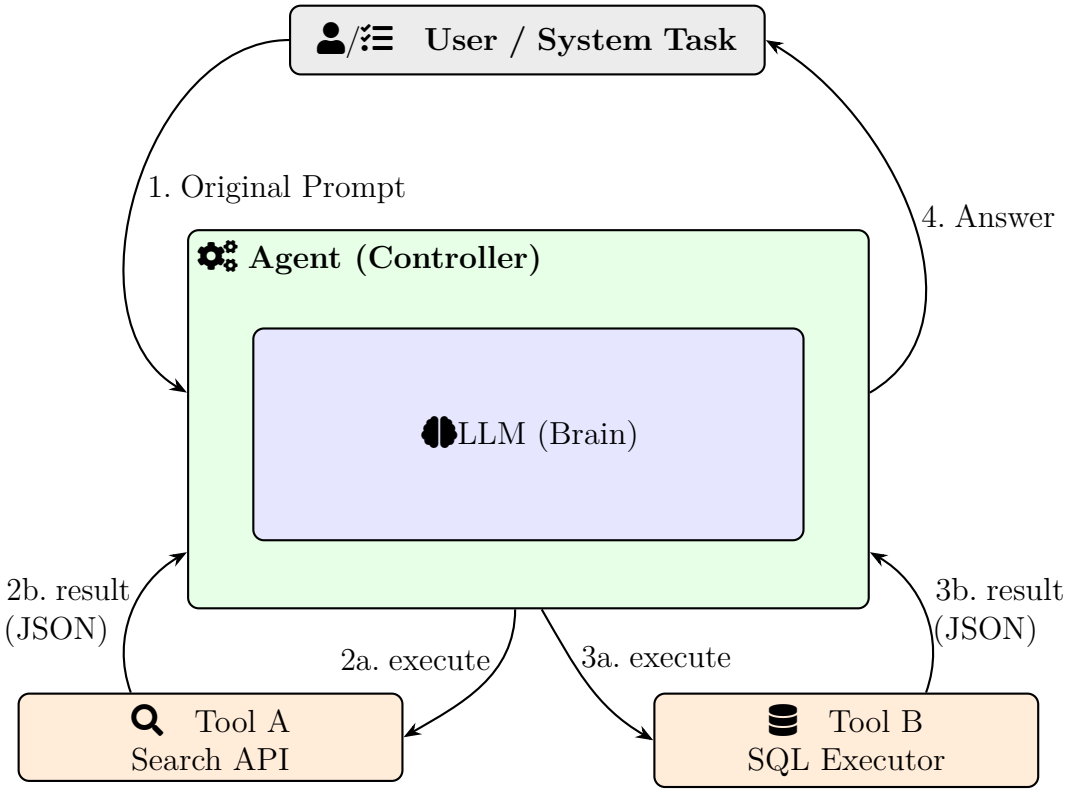


Figure 2.4: Agent-first orchestration with top user/system task source. The task feeds an incoming prompt to the Agent; after iterative reasoning and tool use, the Agent delivers the final answer. The LLM interprets messages, decides next step, and produces structured output. The agent receives prompt, adds tools/policies/context and builds structured messages for the LLM.

the tool (step 3a), receives the result (step 3b, see Listing 2.4b), and appends the resulting data to a new structured message for the LLM. With this enriched context, the LLM reasons one final time and decides to produce the final answer to be generated (see Listing 2.4c). The agent delivers the final answer back to the user/system (step 4).

```sql
SELECT company,
       AVG(close_price) AS avg_price_last_week
FROM stock_prices
WHERE company IN ('NextEra Energy', 'Iberdrola')
  AND trade_date >= CURRENT_DATE - INTERVAL '7 days'
GROUP BY company;
```

Listing 2.3: SQL query generated by the LLM in step 3.

It should be clear from the example of Figure 2.4 that it is not the LLM that directly calls the tool. The LLM's job is to decide that a tool is needed and then generate a structured output (like a JSON object) that describes which tool to

(a) Search API Result

```
{
  "type": "tool_result",
  "tool_name": "Search API",
  "parameters": { "query": "Top 10 renewable energy companies by market
  ↪  capitalization in 2025" },
  "output": [
    { "company": "NextEra Energy", "market_cap": "150B USD" },
    { "company": "Iberdrola", "market_cap": "90B USD" }
  ]
}
```

(b) SQL Executor Result

```
{
  "type": "tool_result",
  "tool_name": "SQL Executor",
  "parameters": {
    "sql_query": "SELECT company, AVG(close_price) ..."
  },
  "output": [
    { "company": "NextEra Energy", "avg_price": 78.52 },
    { "company": "Iberdrola", "avg_price": 12.37 }
  ]
}
```

(c) Final Answer

```
{
  "type": "final_answer",
  "summary": "Average stock prices",
  "data": [
    { "company": "NextEra Energy", "avg_price": 78.52, "currency": "USD" },
    { "company": "Iberdrola", "avg_price": 12.37, "currency": "USD" }
  ],
  "sources": [
    "https://example.com/nextera",
    "https://example.com/iberdrola"
  ]
}
```

Listing 2.4: Structured messages exchanged during an agent execution (e.g., ReAct or Plan-and-Act): (a) Search API result, (b) SQL executor result, (c) final answer produced by the LLM.

use and what arguments to pass to it. The agent (or the orchestration layer in the application) is the component that receives this structured output from the LLM. It then parses that output and actually executes the function or API call associated with the tool. One analogy that can be made is the following: the LLM is the *brain* that reasons and decides; it figures out what needs to be done. The agent is the body that takes the brain's instructions and performs the physical action in the real world. This separation of concerns is fundamental to how tool-calling works in modern LLM applications. It ensures that the LLM, which is a powerful reasoning engine, isn't burdened with the task of execution, while the agent, which is a reliable and predictable executor, can handle the actual interaction with external systems. It's a robust design pattern that maximizes the strengths of both components.

While the example in Figure 2.4 illustrates the general mechanics of tool calling, it abstracts away many of the task-specific details involved in deciding how to use a particular tool. In practice, each type of tool may require its own reasoning strategy, input transformation, and post-processing steps. For instance, generating a search query for an information retrieval module involves different considerations than producing a valid SQL statement for a database engine. The next two paragraphs highlight these differences by previewing how the tool-calling process unfolds in two concrete scenarios that will be explored in detail later in this chapter.

In step 2, the LLM returns to the agent the specification for invoking the Search API, including the textual query "*Top 10 renewable energy companies by market capitalization in 2025*" (Listing 2.4a). Determining this query from the user's original prompt requires retrieving and ranking relevant information, a process that follows the Retrieval-Augmented Generation (RAG) paradigm. The details of this approach will be presented in Section 2.6.

In step 3, the LLM returns to the agent the specification for invoking the SQL Executor, including the query presented in Listing 2.3. Determining this SQL expression is itself a non-trivial process that involves translating natural language into an executable query. The specific techniques and challenges of this Text-to-SQL transformation will be discussed in Section 2.7.

2.6. Retrieval-Augmented Generation

One significant challenge in LLM-based agents is *hallucination*, the tendency of the model to produce outputs that sound plausible but are factually incorrect or unsupported. Hallucinations often arise when the model is asked about information that was likely absent, outdated, or only partially represented in its training data. This includes recent events, niche technical domains, and proprietary datasets. A widely adopted strategy to address this limitation is Retrieval-Augmented Generation (RAG) [Fan et al. 2024], in which the agent retrieves relevant, authoritative information from external sources at query time and feeds it to the LLM. By grounding the model's generation in fresh, domain-specific evidence, RAG both mitigates hallucination and enables the agent to handle queries beyond the scope of its original training corpus.

In practice, RAG works by retrieving relevant documents or passages from a

knowledge base and inserting them directly into the LLM's context window. This integration allows the model to ground its output in concrete, up-to-date evidence, improving factual accuracy and reducing unsupported content. In the earlier tool calling example (Figure 2.4), this corresponds to the Search API step, where the agent formulates a query, retrieves results from an external source, and passes them to the LLM before generating the next reasoning step.

2.6.1. Phases of RAG

A RAG pipeline is typically composed of two phases: *Indexing* and *Retrieval and Generation*. The first phase is executed offline as a form of pre-processing, while the second is performed online, since the language model needs to retrieve relevant information in real time to build context for generating a response. Each of these phases involves several steps, as described below.

Indexing. The steps of this phase are depicted in Figure 2.5. In the *Data Loading* step, documents are collected from targeted sources such as PDF files, web pages, or other forms of unstructured information. In the subsequent *Chunking* step, each document is divided into smaller segments to facilitate processing. Various strategies can be employed for splitting documents, depending on the structure and nature of the document (see Section 2.6.2). Then, each chunk is mapped to a vector representation, using a process usually known as *embedding*. A vector embedding is a numerical representation of text data in a continuous, high-dimensional space that captures semantic or structural similarity. There are several embedding frameworks to map text data to a vector representation, such as BERT, RoBERTa, and BERTimbau. In the last step (*Vector Storage*), these vectors are stored in a *vector database*, which is a specialized data store designed to efficiently index, store, and retrieve high-dimensional vector embeddings based on similarity search. This indexing phase is typically performed offline; afterward, the collection is ready to be queried at runtime so relevant chunks can be injected into the LLM's context window to ground answers.

Retrieval and Generation. The steps of this phase are depicted in Figure 2.6. It starts with *Query Embedding*, where the user's query is transformed into a vector representation using the same embedding model employed during indexing. In *Retrieval*, the system searches the vector database using similarity search (e.g., semantic search or dense retrieval methods such as DPR) to identify the top-k most relevant chunks. These retrieved chunks are then combined with the original query in *Prompt Construction*, forming a context-rich prompt for the language model. Finally, in *Generation*, the LLM processes this augmented prompt to produce a grounded, accurate answer, leveraging both the retrieved evidence and its own reasoning capabilities.

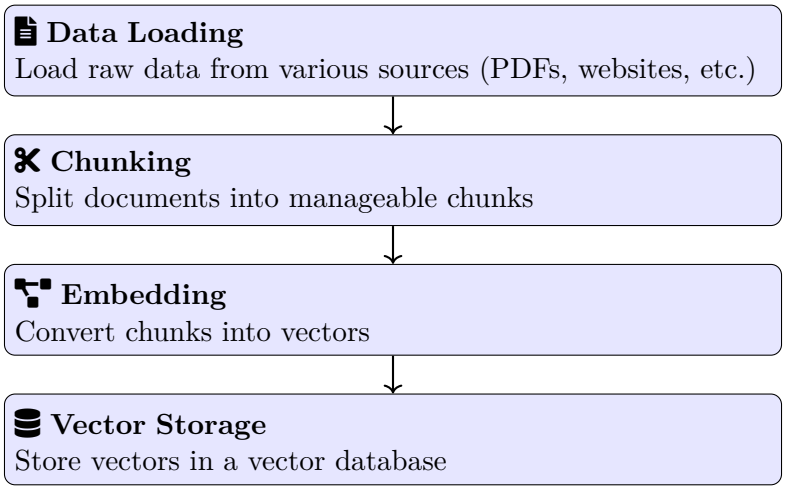


Figure 2.5: Phase 1: Data Indexing (Offline). This phase prepares a knowledge base for use in RAG. It begins with *Data Loading*, where raw data is collected from various sources such as PDFs, websites, or internal documentation. In the *Chunking* step, the documents are split into smaller, manageable segments to improve retrieval granularity. These chunks are then passed through an *Embedding* step, which converts them into high-dimensional vector representations using a pre-trained model. Finally, the resulting vectors are stored in a *Vector Database*, enabling efficient semantic search during the retrieval phase.

2.6.2. Chunking Strategies

In the indexing stage of RAG (see Figure 2.5), the strategy to break each document in the knowledge base into chunks is very important. In a document chunking strategy, several factors directly influence retrieval quality and downstream LLM performance:

1. **Chunk Size Trade-offs**. Chunks that are too small risk losing important context, leading to incomplete or ambiguous matches during retrieval. Chunks that are too large may exceed the model's context limits or dilute relevance by mixing unrelated information. The ideal size balances semantic completeness with retrieval precision, often expressed in tokens or characters.

2. **Overlap Strategy**. Overlapping content between consecutive chunks helps preserve context across chunk boundaries, ensuring that relevant information appearing near the edge of one chunk is still captured in the next. Too much overlap, however, increases storage and retrieval costs without proportional benefits.

3. **Metadata Preservation**. Alongside the chunk text, associated metadata (e.g., source document ID, section headings, timestamps, authorship) should be stored and linked to each chunk. Preserving metadata enables filtered or faceted retrieval, traceability of sources, and better grounding of the final generated response.

There are four commonly used approaches to chunking [Fan et al. 2024,

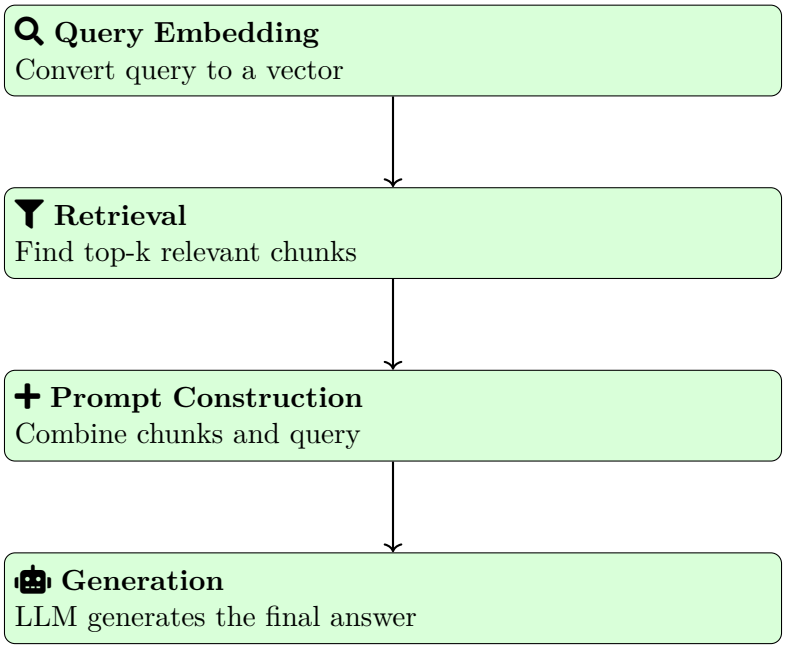


Figure 2.6: Overview of the *Retrieval and Generation* phase in a RAG pipeline. This online phase begins with *Query Embedding*, where the user's natural language query is converted into a dense vector representation. In the *Retrieval* step, this vector is used to perform a similarity search over a vector database to retrieve the top-k most relevant text chunks. These chunks are then combined with the original query during the *Prompt Construction* step to form the context window. Finally, in the *Generation* step, the language model uses this enriched prompt to generate a grounded and context-aware response.

Arslan et al. 2024]: *Fixed-Size Chunking*, *Recursive Character Text Splitting*, *Semantic Chunking*, and *Structure-Based Chunking*. The following paragraphs describe each one of these approaches.

Fixed-size chunking is the most basic approach to splitting text, where the content is divided into consecutive segments of a predetermined size, typically measured in characters or tokens, without regard for linguistic or semantic boundaries. For instance, a document might be split into 500-character chunks with an overlap of 50-100 characters to preserve some context between segments. This method works well for uniformly structured data, such as simple logs, raw transcripts, or other unformatted text streams, but its simplicity comes at a cost: it can interrupt sentences or paragraphs mid-thought, fragmenting ideas and potentially reducing retrieval quality.

Recursive character text splitting is a more sophisticated and widely recommended strategy for general-purpose text, designed to prioritize natural boundaries before falling back to fixed-size segments. It uses a hierarchy of separators, such as paragraph breaks ("\n\n"), line breaks ("\n"), sentence endings ("."), and spaces, to divide the text. The splitter first attempts to segment by paragraphs, then by sentences, and so on; if a chunk remains too large, it defaults to a fixed-size split. This

method is particularly effective for documents with varied structures, like reports, articles, or books, as it preserves paragraphs and sentences intact whenever possible. However, because it still relies on syntactic markers, it may fail to capture semantic relationships that span across these boundaries.

Semantic chunking is an advanced strategy that prioritizes the meaning of the content over its syntactic structure. It begins by breaking the document into smaller units, such as sentences, and then generating vector embeddings for each. By measuring the similarity between embeddings of adjacent sentences, the method detects "semantic breaks" where the similarity falls below a chosen threshold, signaling a shift in topic and prompting the start of a new chunk. This approach is well-suited for unstructured text or content where the flow of ideas takes precedence over rigid formatting, such as conversational exchanges or creative writing. However, it is more computationally demanding and requires both an embedding model and careful tuning of the similarity threshold.

Structure-based chunking leverages the explicit organization of a document to produce meaningful chunks. For example, a Markdown header splitter uses section headers (e.g., "#", "##") as boundaries, creating chunks that align with sections and subsections, often preserving the headers in the metadata to provide additional context. Similarly, an HTML/XML splitter relies on tags to identify logical components such as titles, paragraphs, or lists, ensuring that related content stays together. This method is particularly effective for structured documents like technical manuals or user guides with clear hierarchical organization. Its main limitation is that it is format-dependent and unsuitable for unstructured text.

2.7. Text-to-SQL

Text-to-SQL refers to the process of converting a natural language query into an equivalent SQL statement that can be executed on a relational database [Shi et al. 2025, Liu et al. 2025, Deng et al. 2022]. The objective is to generate a query that faithfully captures the users's intent and returns the correct results when run against the target data. In the context of LLM-based agents, Text-to-SQL serves as a reasoning capability that enables the agent to bridge the gap between free-form language and the structured syntax of SQL, allowing natural language requests to be transformed into precise, executable commands.

In an agent-based system, interaction with a database typically requires that the SQL interface be registered as an available tool, along with its connection details and schema description. Once this registration is in place, the agent can decide when to use the tool in response to a user request. For example, given the question "*What was the total payroll expense for the Sales department last month?*", the agent may determine that answering it requires querying the database. Following the ReAct or Plan-and-Act paradigms (see Section 2.4), the LLM chooses to invoke the SQL interface, using Text-to-SQL reasoning to translate the natural language request into a valid SQL statement. In this way, the database becomes just another tool the agent can call, much like a calculator or search engine, with Text-to-SQL serving as the mechanism that bridges free-form language and precise, executable queries.

Text-to-SQL has long been recognized as a valuable capability in the database community [Shorten et al. 2025], enabling users to query relational databases without writing SQL manually. Typical applications include: (i) business intelligence dashboards with natural language interfaces; (ii) voice or chatbot assistants that retrieve data from corporate systems; and (iii) self-service data exploration tools for non-technical users. In the LLM era, these use cases are increasingly integrated into multi-tool agent frameworks, where Text-to-SQL operates as the bridge between natural language input and executable database queries.

The key steps of a Text-to-SQL pipeline, illustrated in Figure 2.7, describe the sequential reasoning and processing stages required to transform a natural language request into an executable SQL query and a user-friendly answer. While the specific implementation details vary between traditional rule-based systems and modern LLM-powered agents, the underlying process remains similar: interpret the request, map it to the database schema and values, construct the query, execute it, and refine the result as needed. The following sections describe each stage, highlighting common challenges and considerations for LLM-based agents.

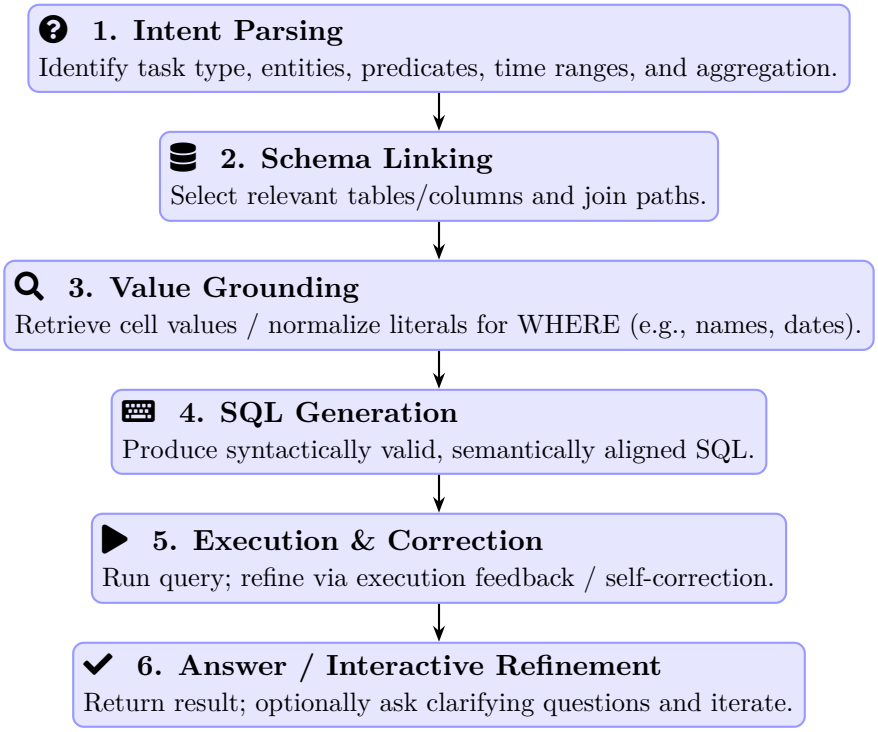


Figure 2.7: Revised Text-to-SQL pipeline. Prior to this pipeline, the SQL interface must be registered/available as a tool (connection + schema).

2.7.1. Intent Parsing

The first step in a Text-to-SQL pipeline is to interpret the user's request and determine its underlying intent. This involves identifying the task type (e.g., retrieval, aggregation, filtering), the key entities mentioned, the attributes of interest, and any constraints such as time ranges, conditions, or grouping requirements.

In LLM-based agents, intent parsing often relies on semantic understanding rather than rigid templates, enabling the model to capture nuanced requests such as "*List the top five customers by total sales in Q2*" or "*How many support tickets were closed last week?*".

Accurate intent parsing ensures that subsequent steps operate with a clear representation of what needs to be retrieved, reducing the risk of generating SQL that is syntactically valid yet semantically misaligned with the user's intent.

2.7.2. Schema Linking

Given a parsed intent, an LLM-based agent maps mentions in the request to concrete schema elements (tables, columns, and relationships) and determines feasible join paths. This typically involves name normalization (singular/plural forms, aliases), leveraging foreign keys and primary keys, and disambiguating homonyms (e.g., `name` in multiple tables) using surrounding context. For large or heterogeneous schemas, the agent often performs schema pruning: serializing a subset of the catalog (table/column summaries, data types, brief descriptions) and including only the most relevant pieces in the context window. Effective schema linking yields a compact schema subgraph that preserves necessary joins and candidate columns while excluding distractors, thereby simplifying subsequent value grounding and SQL generation.

This process is often implemented as a two-stage pipeline: initial retrieval followed by LLM-based validation. For example, consider the prompt "*List the top five customers by total sales in Q2*". The agent first retrieves the top-matching schema items (via embedding search):

- `customers (customer_id, name, region)`
- `orders (order_id, customer_id, order_date, total_amount)`

It then confirms their relevance by sending an intermediary prompt to the LLM, asking which schema items are necessary to answer the question.

Figures 2.8a and 2.8b illustrate how intermediate prompts can be used to confirm schema relevance across different domains. The relevant schema elements extracted by the LLM for each intermediate prompt are summarized in Table 2.3, illustrating how the schema linking step narrows the database context to only the fields necessary for accurate SQL generation.

2.7.3. Value Grounding

The third step in the Text-to-SQL pipeline involves mapping abstract references and natural language expressions in the user query to concrete values that exist in the database schema or data. This includes resolving temporal expressions (e.g., "last month" → `2025-07`, "Q2" → specific date range), performing entity disambiguation (e.g., "Apple" → "Apple Inc." as stored in the `company` table), and normalizing literals (e.g., "Jan" → "January", standardizing currency symbols, abbreviations, or capitalization).

Given this question: “List the top five customers by total sales in Q2” and these candidate schema elements:
- customers: customer_id, name, region - orders: order_id, customer_id, order_date, total_amount
Which elements are necessary to answer the question? Return only the relevant ones.

(a) Sales database example

Given this question: “How many support tickets were closed last week?” and these candidate schema elements:
- tickets: ticket_id, opened_date, closed_date, status, assigned_team - teams: team_id, team_name, department
Which elements are necessary to answer the question? Return only the relevant ones.

(b) Customer support database example

Figure 2.8: Examples of intermediate prompts for the schema linking step. In each case, the agent presents candidate schema elements to the LLM, which identifies the subset needed to answer the question.

Prompt (Figure ref.)	**Relevant schema elements**
Figure 2.8a	`customers: customer_id, name` `orders: customer_id, total_amount`
Figure 2.8b	`tickets: ticket_id, closed_date`

Table 2.3: Relevant schema elements identified by the LLM for each intermediate prompt in the schema linking examples.

Value grounding also encompasses fuzzy matching techniques to handle variations in naming conventions, such as mapping “NYC” to “New York City” or “AI dept” to “Artificial Intelligence Department” as they appear in the database records. In LLM-based agents, this step often combines retrieval mechanisms (used to search for candidate matches in the database) with the model’s reasoning capabilities to infer the most appropriate mappings based on query context.

Effective value grounding is critical for preventing SQL queries from referencing non-existent or incorrectly formatted values in `WHERE` clauses and `JOIN` conditions. By ensuring values are both valid and properly formatted, it increases the likelihood that generated queries will execute successfully and return meaningful, non-empty results rather than failing due to exact-match errors.

The value grounding step is illustrated in Figure 2.9, where entity names and temporal expressions are mapped to concrete database values and date ranges prior

to SQL generation (see Figures 2.9a and 2.9b). In these examples, the mapping assumes the current date is August 2025, so "Q2 last year" corresponds to April 1 to June 30, 2024, and "last month" corresponds to July 1 to July 31, 2025. Such mappings may vary depending on the reference date, calendar conventions, or fiscal year definitions.

Natural language query:
"Show the total sales for Apple in Q2 last year."

Value grounding:
- "Apple" → `"Apple Inc."` (as stored in `company.name`)
- "Q2 last year" → `'2024-04-01'` to `'2024-06-30'`

(a) Entity and temporal grounding with a named company and a quarter-relative period.

Natural language query:
"What was the total payroll expense for the Sales department last month?"

Value grounding:
- "Sales" → `"Sales"` (as stored in `department.name`)
- "last month" → `'2025-07-01'` to `'2025-07-31'`

(b) Department and temporal grounding with a month-relative period.

Figure 2.9: Examples of value grounding: mapping abstract references (entities and time expressions) to concrete database values and ranges before SQL generation.

2.7.4. SQL Generation

The fourth step synthesizes the parsed intent, identified schema elements, and grounded values into a syntactically correct and semantically aligned SQL query. This process involves translating the natural language structure into appropriate SQL constructs, such as mapping aggregation requests to `GROUP BY` clauses with aggregate functions (`COUNT`, `SUM`, `AVG`), temporal filters to `WHERE` conditions with date comparisons, and ranking requirements to `ORDER BY` with `LIMIT` clauses.

The generation must also handle complex query patterns like subqueries for nested conditions, proper `JOIN` syntax to connect multiple tables identified during schema linking, and correct handling of `NULL` values and data types. In LLM-based systems, SQL generation leverages the model's understanding of SQL syntax and semantics, often guided by schema-aware prompting that includes table structures, relationships, and example queries to ensure dialect-specific correctness (e.g., PostgreSQL vs. MySQL syntax differences).

Robust SQL generation produces queries that not only execute without syntax errors but also accurately reflect the user's intent, avoiding common pitfalls such as Cartesian products from missing `JOIN` conditions, incorrect aggregation levels, or inefficient query structures that could cause performance issues on large datasets.

Example 1: Aggregation and Ranking. Consider the prompt: "*List the top five customers by total sales in Q2.*" After **Intent Parsing**, **Schema Linking**, and **Value Grounding**, the system has determined:

- Task type: aggregation + ranking
- Relevant schema: `customers(customer_id, name)`, `orders(customer_id, total_amount, order_date)`
- Grounded values: Q2 = `'2025-04-01'` to `'2025-06-30'`

```sql
SELECT c.name, SUM(o.total_amount) AS total_sales
FROM customers c
JOIN orders o ON c.customer_id = o.customer_id
WHERE o.order_date BETWEEN '2025-04-01' AND '2025-06-30'
GROUP BY c.name
ORDER BY total_sales DESC
LIMIT 5;
```

Listing 2.5: Example SQL generated for the Q2 sales query.

Example 2: Counting Filtered Records. Consider the prompt: "*How many support tickets were closed last month?*" The system has determined:

- Task type: count + filtering by date
- Relevant schema: `tickets(ticket_id, closed_date)`
- Grounded values: last month (August 2025) = `'2025-08-01'` to `'2025-08-31'`

```sql
SELECT COUNT(*) AS closed_tickets
FROM tickets
WHERE closed_date BETWEEN '2025-08-01' AND '2025-08-31';
```

Listing 2.6: Example SQL generated for counting closed support tickets.

2.7.5. Execution and Correction

The fifth step is to execute the generated SQL query against the target database and address any errors that occur. This step forms a critical feedback loop in the Text-to-SQL pipeline. The agent submits the query and receives either a valid result set or an error message from the database engine. If an error is returned (e.g., syntax violation, invalid column name, missing table, or permission restriction), the system must interpret the message and use it to guide a correction.

In LLM-based agents, the error output is typically fed back into the model as an observation. The LLM then analyzes the failure, identifies its likely cause (such as a misspelled column name, mismatched data type, or missing `JOIN` condition), and produces a revised query. This iterative execution-correction cycle continues until a valid query is obtained or a stopping condition is reached. By incorporating this self-repair mechanism, the LLM shifts from being a static code generator to a dynamic, resilient problem-solver capable of adapting to schema-specific constraints and recovering from its own mistakes.

The interaction between SQL generation, execution, and error-driven correction is illustrated in Figure 2.10, which shows how an LLM-based agent iteratively refines queries until a valid result is produced.

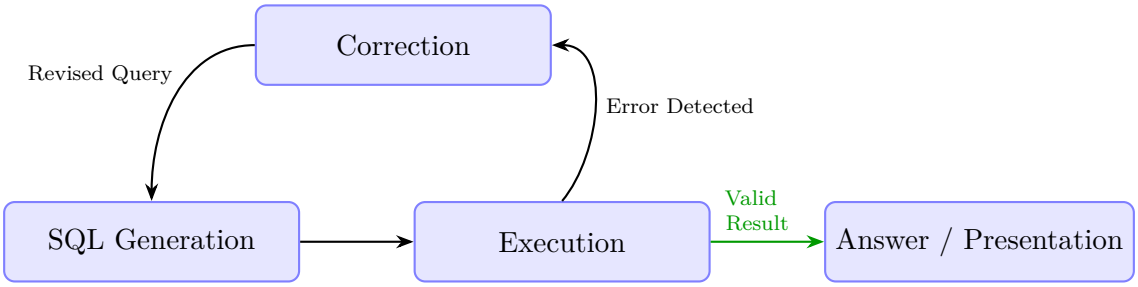


Figure 2.10: Execution and correction loop. Successful execution yields a valid result that flows to answer presentation; errors trigger correction and a revised query that feeds back into SQL generation.

2.7.6. Answer and Interactive Refinement

The final step in the Text-to-SQL pipeline is to present the query result to the user in a clear, natural language format and, when needed, to engage in a dialogue for further refinement. The raw result set returned by the database, typically a table of rows and columns, is often not user-friendly. The agent's role is to interpret this data and produce a concise, human-readable answer that directly addresses the original request. For instance, a result containing a single count might be presented as: "There were 15 support tickets closed last week," rather than displaying a one-row, one-column table.

If the initial request was ambiguous, incomplete, or yielded unexpected results, this step also enables an interactive refinement process. The user can issue follow-up queries (e.g., "*What about the week before that?*") or clarify intent (e.g., "*No, I meant closed by the engineering team*"). The system can then restart the pipeline from an appropriate earlier stage, such as intent parsing or value grounding, while incorporating the additional context, ultimately generating a more precise and satisfactory answer.

This process is illustrated in Figure 2.10, where the agent's response generation is not the end of the process but part of an interactive cycle that can loop back to earlier stages when clarification or refinement is needed.

Figure 2.11 demonstrates how an agent transforms raw database outputs into

concise, user-friendly answers and supports iterative refinement based on follow-up questions.

count
15

Agent: There were 15 support tickets closed last week.

(a) Initial query result transformed from raw SQL output to a clear natural language answer.

count
12

User: What about the week before that?
Agent: There were 12 support tickets closed in the previous week.

(b) Interactive refinement after the initial answer.

Figure 2.11: Examples of the answer and interactive refinement step: (a) raw SQL output reformulated as natural language; (b) follow-up interaction to refine the result.

The process of transforming raw SQL results into a concise, user-friendly response is illustrated in Figure 2.12. In this case, the query result is a single count value (15), which, following the approach in Figure 2.11a, is reformulated into the natural language answer "*There were 15 support tickets closed last week*". This example demonstrates how the agent bridges the gap between structured data and conversational output, ensuring that the final response directly addresses the user's original question.

The user asked: "How many support tickets were closed last week?"
The SQL query returned this result:
count
15
Respond with a natural language sentence directly answering the question.

Figure 2.12: Example of a prompt guiding the agent to transform a raw SQL result into a natural language response.

2.8. Case Studies and Demonstrations

To consolidate the concepts presented throughout this chapter, we provide a collection of Jupyter notebooks that illustrate the main topics covered, from the foundations of large language models to the design and implementation of LLM-based agents with multi-step reasoning and tool use. These notebooks are available

at `https://github.com/AILAB-CEFET-RJ/sbbd2025_course`), and are designed to be self-contained and reproducible, enabling readers to experiment directly with the techniques, architectures, and workflows described in the text. The demonstrations include:

1. *From Statistical Models to LLMs.* A hands-on comparison between n-gram language models and small transformer-based models, illustrating differences in context windows, scaling, and emergent capabilities.

2. *From LLMs to LLM-based Agents.* Construction of a minimal agent that uses an LLM as its reasoning core, demonstrating perception, decision-making, and tool invocation in simple structured tasks.

3. *Prompting Techniques and Interaction Patterns.* Experiments with zero-shot, few-shot, and Chain-of-Thought prompting, followed by implementations of interaction patterns such as ReAct, Plan-and-Act, and Pre-Act.

4. *Tool Calling.* Step-by-step examples of tool registration, structured message exchange, and execution via APIs, search engines, and SQL interfaces.

5. *Retrieval-Augmented Generation (RAG).* Building a complete RAG pipeline, including document chunking, embedding generation, vector storage, retrieval, and grounded generation.

6. *Text-to-SQL Pipeline.* An end-to-end example covering intent parsing, schema linking, value grounding, SQL generation, execution and correction, and answer presentation with interactive refinement.

Each notebook follows a consistent structure: defining the required data and tools, walking through each relevant pipeline stage, and allowing readers to modify prompts, parameters, and components to observe their effect. By running these notebooks, readers can explore how the concepts described in the chapter translate into working implementations, reinforcing theoretical understanding with hands-on practice.

2.9. Final Remarks

This chapter provided an introduction to LLM-based agents, tracing their evolution from statistical language models to modern, tool-using architectures capable of reasoning and acting in complex environments. We examined core building blocks such as prompting techniques, interaction patterns, tool calling, retrieval-augmented generation, and the Text-to-SQL pipeline, illustrating how these components integrate into coherent, end-to-end workflows. Beyond theory, the accompanying Jupyter notebooks provide practical demonstrations that enable readers to experiment with these ideas in real-world scenarios. Together, these elements aim to equip the reader with both the conceptual understanding and the hands-on skills necessary to design, implement, and critically evaluate LLM-based agents in diverse application domains.

Looking ahead, the field is poised to advance rapidly along several fronts. Emerging trends such as multi-agent systems (agent collaboration), seamless multimodality, dynamic memory management, and domain-specific fine-tuning will expand both the capabilities and the applicability of LLM-based agents. At the same time, significant challenges remain, including ensuring factual reliability, safeguarding data privacy, maintaining transparency in decision-making, and mitigating bias amplification. Mitigation strategies include grounding outputs in authoritative data (e.g., via RAG), maintaining detailed logs of reasoning steps and tool calls for auditability, applying domain-specific access controls, and performing bias and safety evaluations during development. Embedding such safeguards into the design and operation of agents is crucial to align technical capabilities with legal requirements, organizational policies, and societal expectations. Addressing these challenges will require not only technical innovation but also interdisciplinary collaboration between AI researchers, domain experts, and policymakers. By combining robust architectures with responsible deployment practices, the next generation of LLM-based agents can evolve from promising prototypes to dependable tools that operate safely and effectively in high-impact real-world settings.

Finally, it is worth noting that the quest to build intelligent agents is far from new. As Jennings and Wooldridge observed more than two decades ago [Jennings and Wooldridge 1998], agents are autonomous problem-solving entities capable of operating in complex, dynamic environments without continuous human guidance. While the technological substrate has shifted (from symbolic reasoning systems to large-scale neural models), the essence of this vision remains strikingly relevant. Today's LLM-based agents embody many of the same aspirations outlined in that earlier work, demonstrating that the principles articulated then continue to guide and inspire the development of the next generation of AI systems.

Acknowledgments

This chapter was developed with the assistance of artificial intelligence tools, which were used to support tasks such as drafting, revising, and refining text, as well as suggesting examples and clarifying technical concepts. All content was critically reviewed and edited by the author to ensure accuracy, coherence, and alignment with the intended objectives.

References

[Anderson 1972] Anderson, P. W. (1972). More is different. *Science*, 177(4047):393–396.

[Arslan et al. 2024] Arslan, M., Ghanem, H., Munawar, S., and Cruz, C. (2024). A survey on RAG with llms. *Procedia Computer Science*, 246:3781–3790. 28th International Conference on Knowledge Based and Intelligent information and Engineering Systems (KES 2024).

[Bengio et al. 2003] Bengio, Y., Ducharme, R., Vincent, P., and Jauvin, C. (2003). A neural probabilistic language model. *Journal of Machine Learning Research*,

3:1137–1155.

[Bezerra 2016] Bezerra, E. (2016). Introdução à aprendizagem profunda. In Ogasawara, V., editor, *Tópicos em Gerenciamento de Dados e Informações*, chapter 3, pages 57–86. SBC, Porto Alegre, Brazil, 1 edition.

[Deng et al. 2022] Deng, N., Chen, Y., and Zhang, Y. (2022). Recent advances in text-to-SQL: A survey of what we have and what we expect. In *COLING*, Gyeongju, Republic of Korea. International Committee on Computational Linguistics.

[Devlin et al. 2019] Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. (2019). BERT: Pre-training of deep bidirectional transformers for language understanding. In Burstein, J., Doran, C., and Solorio, T., editors, *ACL 2019*, pages 4171–4186, Minneapolis, Minnesota. ACL.

[Erdogan et al. 2025] Erdogan, L. E., Furuta, H., Kim, S., et al. (2025). Plan-and-act: Improving planning of agents for long-horizon tasks. In *ICML 2025*.

[Fan et al. 2024] Fan, W., Ding, Y., Ning, L., Wang, S., Li, H., Yin, D., Chua, T.-S., and Li, Q. (2024). A survey on RAG meeting llms: Towards retrieval-augmented large language models. In *KDD'24*, KDD'24, page 6491–6501, New York, NY, USA. Association for Computing Machinery.

[Hu et al. 2025] Hu, S., Kim, S. R., Zhang, Z., et al. (2025). Pre-act: Multi-step planning and reasoning improves acting in LLM agents. *arXiv preprint arXiv:2505.09970*.

[Jennings and Wooldridge 1998] Jennings, N. R. and Wooldridge, M. J., editors (1998). *Agent Technology: Foundations, Applications, and Markets*. Springer, Berlin, Heidelberg.

[Kayhan et al. 2023] Kayhan, V., Levine, S., Nanda, N., Schaeffer, R., Natarajan, A., Chughtai, B., et al. (2023). Scaling laws and emergent capabilities of large language models. *arXiv preprint arXiv:2309.00071*.

[LeCun et al. 2015] LeCun, Y., Bengio, Y., and Hinton, G. (2015). Deep learning. *Nature*, 521(7553):436–444.

[Liu et al. 2025] Liu, X., Shen, S., Li, B., Ma, P., Jiang, R., Zhang, Y., Fan, J., Li, G., Tang, N., and Luo, Y. (2025). A survey of text-to-sql in the era of llms: Where are we, and where are we going? *IEEE Transactions on Knowledge and Data Engineering*, pages 1–20.

[Mikolov et al. 2010] Mikolov, T., Karafiát, M., Burget, L., Černockỳ, J., and Khudanpur, S. (2010). Recurrent neural network based language model. In *INTERSPEECH*, pages 1045–1048.

[Newell et al. 1956] Newell, A., Shaw, J., and Simon, H. A. (1956). The logic theory machine–a complex information processing system. *IRE Transactions on Information Theory*, 2(3):61–79.

[Nilsson 1984] Nilsson, N. J. (1984). *Shakey the Robot*. SRI International, Menlo Park, CA.

[Rawat et al. 2025] Rawat, M., Gupta, A., et al. (2025). Pre-act: Multi-step planning and reasoning improves acting in llm agents. *arXiv preprint arXiv:2505.09970*.

[Rosenfeld 2000] Rosenfeld, R. (2000). Two decades of statistical language modeling: where do we go from here? *Proceedings of the IEEE*, 88(8):1270–1278.

[Russell and Norvig 2021] Russell, S. and Norvig, P. (2021). *Artificial Intelligence: A Modern Approach*. Pearson, 4th edition.

[Sapkota et al. 2025] Sapkota, R., Roumeliotis, K. I., and Karkee, M. (2025). Ai agents vs. agentic ai: A conceptual taxonomy, applications and challenges.

[Shi et al. 2025] Shi, L., Tang, Z., Zhang, N., Zhang, X., and Yang, Z. (2025). A survey on employing large language models for text-to-sql tasks. *ACM Comput. Surv.*

[Shorten et al. 2025] Shorten, C., Pierse, C., Smith, T. B., D'Oosterlinck, K., Celik, T., Cardenas, E., Monigatti, L., Hasan, M. S., Schmuhl, E., Williams, D., Kesiraju, A., and van Luijt, B. (2025). Querying databases with function calling.

[Vaswani et al. 2017] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. (2017). Attention is all you need. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 5998–6008.

[Wei et al. 2022a] Wei, J., Tay, Y., Bommasani, R., Raffel, C., Zoph, B., Borgeaud, S., Yogatama, D., Bosma, M., Zhou, D., Metzler, D., Chi, E. H., Hashimoto, T. B., Vinyals, O., Liang, P., Dean, J., and Fedus, W. (2022a). Emergent abilities of large language models. *arXiv preprint arXiv:2206.07682*.

[Wei et al. 2022b] Wei, J., Wang, X., Schuurmans, D., Bosma, M., Ichter, B., Xia, F., Chi, E. H., Le, Q. V., and Zhou, D. (2022b). Chain-of-thought prompting elicits reasoning in large language models. *arXiv preprint arXiv:2201.11903*.

[Yao et al. 2023] Yao, S., Yu, D., Zhao, J., Shafran, I., Griffiths, T. L., Cao, C., and Narasimhan, K. (2023). Tree of thoughts: Deliberate problem solving with large language models. *arXiv preprint arXiv:2305.10601*.

[Yao et al. 2022] Yao, S., Zhao, J., Yu, D., Du, N., Yu, W.-t., Shafran, I., Griffiths, T. L., Neubig, G., Cao, C., and Narasimhan, K. (2022). React: Synergizing reasoning and acting in language models. *arXiv preprint arXiv:2210.03629*.