arXiv:2409.17120v2 [cs.CL] 8 Dec 2025



Deep Learning and Machine Learning: Advancing Big Data Analytics and Management

Benji Peng[1*], Xuanhe Pan[2*], Yizhu Wen[3*], Ziqian Bi[4*], Keyu Chen[5], Ming Li[5], Ming Liu[6], Qian Niu[7], Junyu Liu[7], Jinlang Wang[2], Sen Zhang[8], Jiawei Xu[6], Xinyuan Song[10], Zekun Jiang[11], Tianyang Wang[12], Pohsun Feng[9†]

[1]AppCubic, [2]University of Wisconsin-Madison, [3]Department of Electrical and Computer Engineering, University of Hawaii at Manoa, [4]Indiana University, [5]Georgia Institute of Technology, [6]Purdue University, [7]Kyoto University, [8]Rutgers University, [9]National Taiwan Normal University, [10]Emory University, [11]Sichuan University, [12]XiâĂŹan Jiaotong-Liverpool University

*Equal contribution

†Corresponding author, 41075018h@ntnu.edu.tw

Contents

0.1 Introduction . . . 7
0.1.1 Simplifying Deep Learning: From Complex to Simple . . . 7
0.1.2 Applications of Machine Learning and Deep Learning . . . 7
Transformer . . . 7
ChatGPT and Generative Pre-trained Models (GPT) . . . 8
XGBoost (eXtreme Gradient Boosting) . . . 9
AlphaGo Zero . . . 9
ResNet (Residual Networks) . . . 10
BERT (Bidirectional Encoder Representations from Transformers) . . . 10
YOLO (You Only Look Once) . . . 10
AlphaFold . . . 11
GAN (Generative Adversarial Networks) . . . 11
Reinforcement Learning . . . 12
0.1.3 Pre-trained Models: Enhancing Model Performance and Accuracy . . . 12
The Concept of Pre-trained Models . . . 12
Common Pre-trained Models . . . 12
Advantages of Pre-trained Models . . . 12
Future Developments of Pre-trained Models . . . 13
How to Use Pre-trained Models in Applications . . . 13
0.1.4 Big Data Management and Processing Technologies . . . 13
SQL and NoSQL Databases . . . 13
SQL Databases . . . 13
NoSQL Databases . . . 14
Graph Databases and Relational Data Analysis . . . 14
Graph Databases . . . 14
Cluster Computing and Distributed Systems . . . 14
Apache Hadoop . . . 14
Apache Spark . . . 14
0.1.5 Future Competitiveness: Essential Skills in Deep Learning and Data Management 14
0.1.6 Goals of This Book . . . 15
Simplifying Complex Technologies to Lower the Learning Curve . . . 15
Comprehensive Coverage of Applications and Classic Models . . . 15
Core Data Management Technologies and Tools . . . 15
Visualization and Practice-Driven Learning . . . 15
A Comprehensive and Systematic Learning Path . . . 16

I Handy Appetizer 17

1 Getting Started with Python 19
1.1 Why Python? . . . 19
1.1.1 What is Python Used For? . . . 19
1.2 What is an IDE? . . . 19
1.3 Setting Up Python . . . 19
1.3.1 Step 1: Download Python . . . 19
1.3.2 Step 2: Install Python . . . 20
1.4 Step 3: Using IDLE - Your First Python IDE . . . 21
1.5 Your First Python Program . . . 21
1.5.1 Step-by-Step Instructions . . . 22
1.6 Quick Recap . . . 22

2 Understanding Python Packages and `pip` 23
2.1 What are Python Packages? . . . 23
2.2 Why Do We Need Packages? . . . 23
2.3 How to Use Packages in Python . . . 23
2.4 What is `pip`? . . . 23
2.5 Why Do We Need `pip`? . . . 24
2.6 Example: Installing `seaborn` Using `pip` . . . 24
2.7 Use `seaborn` to plot . . . 24
2.7.1 Example: Using `seaborn` to Create a Simple Plot . . . 24
2.7.2 Another Example: Visualizing the `tips` Dataset . . . 26
2.8 Summary . . . 27

3 Machine Learning 101 29
3.1 What is Machine Learning? . . . 29
3.2 What Can Machine Learning Do? . . . 29
3.3 Understanding Machine Learning . . . 29
3.4 Introducing `scikit-learn` . . . 30
3.4.1 Installing `scikit-learn` . . . 30
3.5 Training a Linear Regression Model on the Diabetes Dataset . . . 30
3.5.1 Step 1: Load the Diabetes Dataset and Train a Linear Regression Model . . . 30
3.6 Visualizing the Result . . . 31
3.7 Training a Logistic Regression Model on the Breast Cancer Dataset . . . 32
3.7.1 Load the Breast Cancer Dataset and Train a Logistic Regression Model . . . 32
3.8 Using ChatGPT to Generate Machine Learning Code . . . 34
3.8.1 How to Prompt ChatGPT for Code . . . 35
3.8.2 Example Response from ChatGPT . . . 35
3.9 Summary . . . 35

4 Deep Learning 101 37
4.1 What is Deep Learning? . . . 37
4.2 What Can Deep Learning Do? . . . 37
4.3 How Does Deep Learning Work? . . . 37
4.4 Mainstream Deep Learning Frameworks . . . 38

4.5 Introducing `tensorflow` (TensorFlow 2) . . . 39
4.5.1 Installing `tensorflow` . . . 40
4.6 Training a Neural Network on the MNIST Dataset . . . 40
4.6.1 Step-by-Step: Training a Neural Network with TensorFlow 2 . . . 40
4.7 Training Output and Accuracy . . . 41
4.8 Introducing `torch` (PyTorch) . . . 41
4.8.1 Installing `torch` . . . 42
4.9 Training a Neural Network on the MNIST Dataset . . . 42
4.9.1 Step-by-Step: Training a Neural Network with PyTorch . . . 42
4.10 Understanding Neural Networks (Simplified) . . . 43
4.11 Using ChatGPT to Generate Deep Learning Code . . . 44
4.11.1 How to Prompt ChatGPT for Code . . . 44
TensorFlow 2 Version . . . 44
PyTorch Version . . . 45
4.12 Summary . . . 46

5 Cool AI Applications That Are Changing the Game 47
5.1 Cutting-Edge Computer Vision (CV) Tools . . . 48
5.1.1 Segment Anything Model (SAM) . . . 48
5.1.2 Digital Avatars . . . 48
5.2 AI in Natural Language Processing (NLP) . . . 49
5.2.1 ChatGPT . . . 49
5.2.2 Gemini: The AI Personal Assistant . . . 50
5.3 AI for Creative Industries . . . 50
5.3.1 Flux: AI for Photography . . . 50
5.3.2 AI Coding Tools: The Future of Software Development . . . 51
5.4 AI in Geospatial Intelligence . . . 52
5.4.1 Geospy.ai: Geospatial AI . . . 52
5.5 Summary . . . 53

6 An Introduction to Generative art: A Beginner's Exploration 55
6.1 What are Diffusion models? . . . 55
6.2 Stable Diffusion: A Game-Changer in AI Art Generation . . . 55
6.3 AI Art: The Fusion of Creativity and Technology . . . 57
6.4 Two Main Interfaces: WebUI vs. ComfyUI . . . 58
6.4.1 WebUI: The User-Friendly Option . . . 58
6.4.2 ComfyUI: Advanced Workflow Customization . . . 58
6.5 Comparison of WebUI and ComfyUI . . . 59
6.6 Getting Started with WebUI . . . 59
6.6.1 Installation . . . 60
6.6.2 Generate Your First Image . . . 60
6.7 Getting Started with ComfyUI . . . 62
6.7.1 Installation . . . 62
Try ComfyUI in Google Colab . . . 64
6.7.2 Install ComfyUI Manager . . . 64
6.7.3 Setting Up Your Default Workflow . . . 65

6.7.4 Generate Your First Image 66
6.7.5 Try Other Models 67
6.7.6 Advanced Features 68
6.8 Summary 69

A Image Demos **71**

0.1 Introduction

In the past decade, the rapid development of Artificial Intelligence (AI), Machine Learning (ML), and Deep Learning (DL) technologies has fundamentally transformed numerous industries, driving global innovation and productivity to new heights [1]. Data has become the core asset of modern enterprises and organizations [2], and the ability to extract valuable insights and knowledge from vast amounts of data has become a key competitive advantage [3]. With the widespread use of big data, machine learning, and deep learning have become essential technologies for automating processes, optimizing decisions, and improving efficiency [4, 5, 6].

Although deep learning and machine learning have tremendous potential, many people find these technologies complex and difficult to understand. This book aims to simplify these complex mathematical formulations, provide intuitive visualizations, and provide comprehensive explanations of core concepts of deep learning and machine learning, allowing readers to easily grasp these technologies and understand their wide-ranging applications [7].

0.1.1 Simplifying Deep Learning: From Complex to Simple

Deep learning (DL) technology based on artificial neural networks typically involves complex mathematical computations such as linear algebra, calculus, and probability theory. These technical foundations have contributed to the success of deep learning, but also present a barrier for many beginners.

One of the core goals of this book is to simplify these complex concepts. We will use simplified mathematical derivations, practical example codes, and graphical demonstrations to make the key concepts of deep learning easier to understand. Using visualization techniques, readers will be able to intuitively grasp the workings of forward propagation and backpropagation in neural networks, how convolutional neural networks (CNNs) [8] operate, and how model parameters are optimized using gradient descent.

In addition, this book will provide practical projects and case studies to help readers understand how these technologies can be applied in real-world scenarios. This approach will not only help technical professionals master deep learning but will also provide business managers with insights into how to leverage these technologies in the era of big data.

0.1.2 Applications of Machine Learning and Deep Learning

Machine learning and deep learning have become the foundation of modern technology systems, driving innovation and efficiency across industries. In the following, we will introduce several classic models and technologies that have played an irreplaceable role in various fields. The models and technologies that we introduce are not limited to these, but these represent some of the most widely applied and impactful models in the industry.

Transformer

The Transformer is a deep learning architecture [9] based on self-attention mechanisms (Figure 1 from [9]), originally designed for Natural Language Processing (NLP) but now extended to image processing and speech recognition. Unlike traditional sequence models such as RNNs [10] and LSTMs [11], the Transformer can process input sequences in parallel, making it more efficient for handling long texts or complex sequences. The Transformer architecture [9] is built on models like GPT [12] and

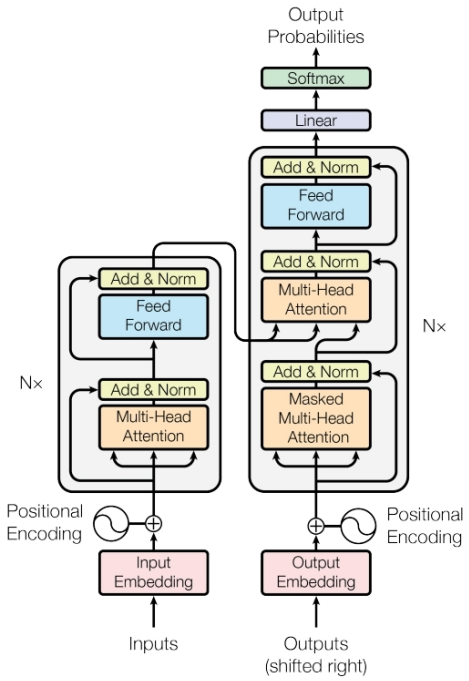
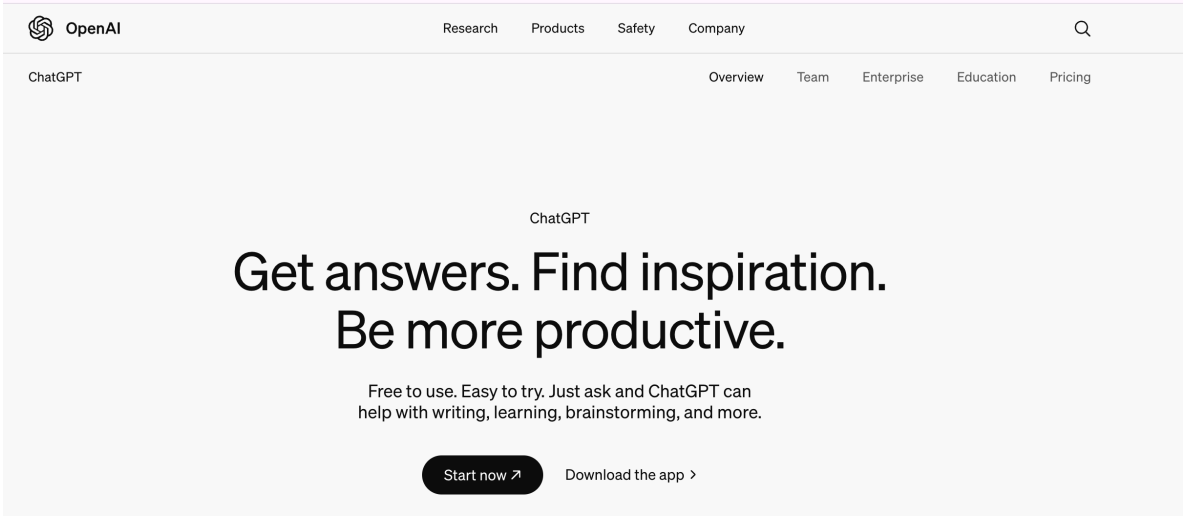


Figure 1: Transformer Model Architecture

BERT [13], used widely in NLP applications such as language translation, text generation, and question answering [14].

ChatGPT and Generative Pre-trained Models (GPT)



ChatGPT is based on OpenAIâĂŹs Generative Pre-trained Transformer (GPT) [15], trained on vast amounts of text data, capable of generating coherent, natural conversations. It has been widely applied in chatbots, text generation, and virtual assistants [16]. The GPT series of models has become an essential technology in natural language processing, pushing the boundaries of language models in content generation, automated customer service, and writing assistance.

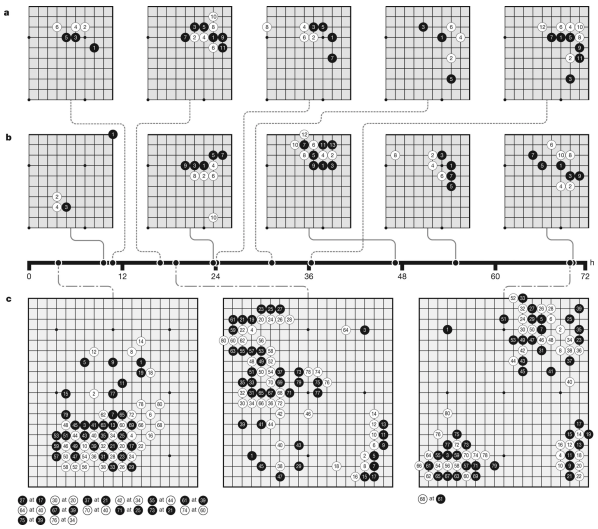
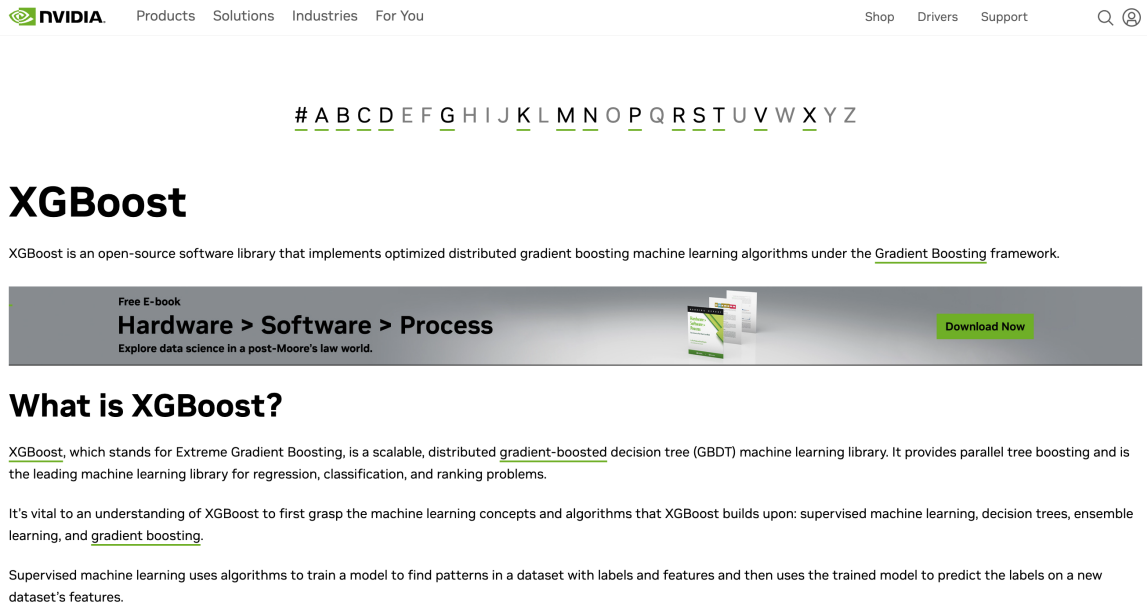


Figure 2: AlphaGo Zero

XGBoost (eXtreme Gradient Boosting)



XGBoost [17] is a decision-tree-based ensemble machine learning model that is particularly suited for handling structured data. Its efficiency and accuracy have made it a popular tool, especially in Kaggle competitions. A wide range of industries, including finance, healthcare, and marketing, use XGBoost to analyze structured data, making it indispensable [18, 19].

AlphaGo Zero

AlphaGo Zero [20] is an AI system based on deep reinforcement learning, which learns complex Go strategies through self-play. Unlike earlier AlphaGo versions, AlphaGo Zero (Figure 2 from [20]) continuously improves through self-competition, eventually surpassing humans. It has applications in autonomous driving, robotic control, and more, and it illustrates the immense potential of deep reinforcement learning for complex decision-making and automation.

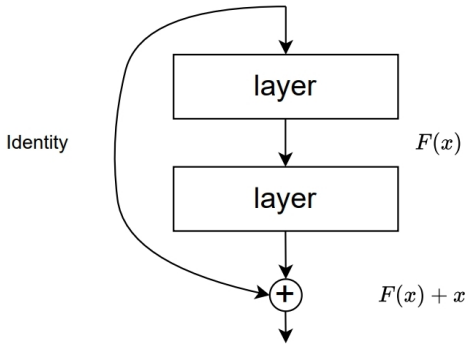


Figure 3: Residual Networks Architecture

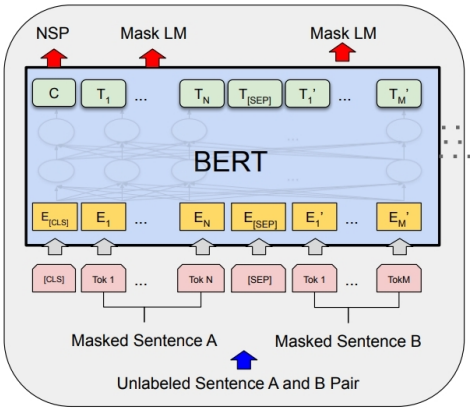


Figure 4: BERT Pre-training Architecture

ResNet (Residual Networks)

ResNet [21] is a specific convolutional neural network model, that addresses the vanishing gradient problem in deep networks through residual connections (Figure 3), enabling much deeper networks to be trained. It is widely applied in medical image analysis, autonomous driving, and visual processing in autonomous vehicles. ResNet is capable of analyzing images, detecting objects, and classifying them.

BERT (Bidirectional Encoder Representations from Transformers)

BERT, developed by Google [22], is a bidirectional encoder model that excels at understanding the contextual semantics of words in text (Figure 4 from [22]). In tasks like sentiment analysis, question answering, and text classification, BERT has become a benchmark technology. Unlike GPTâĂŹs unidirectional generative model, BERTâĂŹs bidirectional encoding captures richer contextual information, providing powerful support for various NLP tasks.

YOLO (You Only Look Once)

YOLO [23, 24] is a real-time object detection model that can locate and classify multiple objects in a single image (Figure 5 from [23]). Due to its outstanding speed and accuracy, YOLO is widely used in real-time detection applications such as autonomous driving, security surveillance, smart cities, and industrial manufacturing.

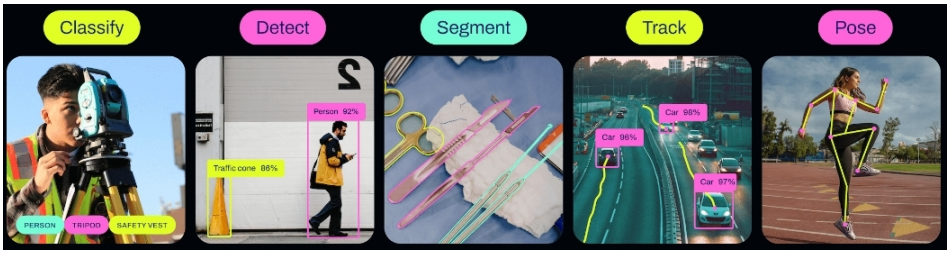


Figure 5: YOLO Models



Figure 6: Q8W3K0: A potential plant disease resistance protein. Mean pLDDT 82.24.

AlphaFold

AlphaFold [25] is a deep learning model developed by Google DeepMind that predicts the three-dimensional structure of proteins (Figure 6 from [26]). This model solves a major challenge in biology, significantly accelerating drug discovery and disease research, and bringing revolutionary changes to life sciences and biotechnology.

GAN (Generative Adversarial Networks)

Generative Adversarial Networks (GANs) [27, 28] consists of a generator and a discriminator (Figure7 from [29]), where the generator creates data and the discriminator distinguishes between real and fake data. GANs are widely used in image generation, style transfer, image restoration, and artistic creation, and can generate realistic images, videos, and audio, greatly expanding the applications of deep learning in content creation.



Figure 7: GAN Mechanism

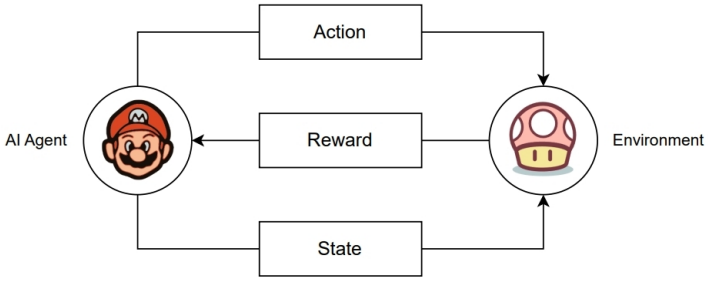


Figure 8: Reinforcement Learning

Reinforcement Learning

Reinforcement [30, 31, 32, 33] learning is a machine learning method focused on learning optimal behaviors through interactions with the environment (Figure 8). Through trial and error, reinforcement learning optimizes decision-making. It has shown great potential in fields such as autonomous driving, robotic control, financial market optimization, and resource management.

0.1.3 Pre-trained Models: Enhancing Model Performance and Accuracy

The Concept of Pre-trained Models

Pre-trained models [34] are trained on large datasets before being applied to specific tasks. In the development of deep learning, pre-trained models have significantly reduced training time while improving performance and accuracy on tasks with limited data.

Common Pre-trained Models

- **BERT (Bidirectional Encoder Representations from Transformers):** Used in natural language processing, BERT is a bidirectional model that considers the context of a sentence from both directions to understand semantics. Pre-trained on massive text datasets, BERT is well-suited for various language tasks, such as question-answering systems and language translation.
- **GPT (Generative Pre-trained Transformer):** GPT models are generative models used to produce text and understand context, widely applied in conversation generation and content creation. ChatGPT is one such application of this technology.
- **ResNet (Residual Networks):** ResNet is a pre-trained model used for image recognition. By introducing residual structures, it solves the degradation problem in deep neural networks. After being pre-trained on large-scale image datasets, ResNet can be fine-tuned for specific image recognition tasks.

Advantages of Pre-trained Models

- **Improved Learning Efficiency:** Pre-trained models have already learned general features from large datasets, allowing them to achieve high performance on new tasks with minimal fine-tuning.
- **Reduced Computational Cost:** Compared to training models from scratch, using pre-trained models saves significant computational resources and time.

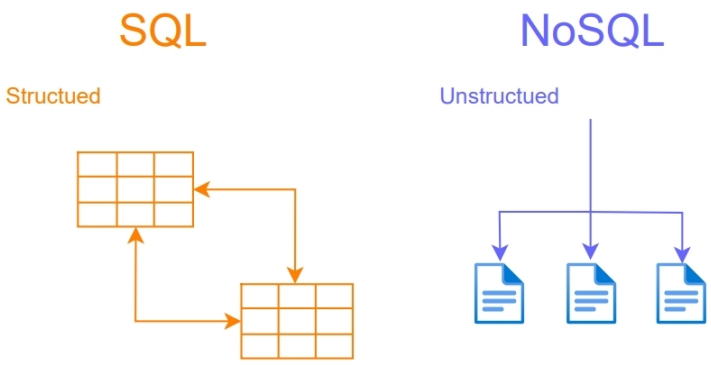


Figure 9: SQL and NoSQL Database

- **Wide Range of Applications:** Pre-trained models are widely applied across various fields, from natural language processing and image recognition to medical image analysis and autonomous driving.

Future Developments of Pre-trained Models

With the continuous expansion of dataset sizes and improvements in computational power, pre-trained models will play a larger role in more fields, and more efficient architectures are likely to emerge, such as models capable of handling multi-modal data.

How to Use Pre-trained Models in Applications

In practice, the main methods for using pre-trained models include:

1. **Transfer Learning:** Fine-tuning pre-trained weights for specific tasks.
2. **Feature Extraction:** Using intermediate layers of pre-trained models to extract features from data and training custom models with these features.

0.1.4 Big Data Management and Processing Technologies

The success of deep learning and machine learning is inseparable from the support of vast amounts of data. However, effectively storing, managing, and processing this data presents a significant challenge. This book will introduce the key technologies of modern data management, helping readers understand how these tools provide the necessary support for machine learning and deep learning models.

SQL and NoSQL Databases

SQL Databases Traditional relational databases (such as MySQL, PostgreSQL) are well-suited for handling structured data [35, 36] (see the left side of Figure 9), especially in scenarios that require transactional consistency and strict data management. We will demonstrate how to use SQL databases for data storage and queries and their applications in industries like finance and retail.

NoSQL Databases To cope with the explosive growth of unstructured data, NoSQL databases [37] (such as MongoDB, Cassandra) provide flexible solutions for storing and processing large-scale data (see the right side of Figure 9). NoSQL databases are widely applied in scenarios like IoT and social media data analysis.

Graph Databases and Relational Data Analysis

Graph Databases Graph databases [38] (such as Neo4j) efficiently store and query complex relationship networks among nodes and edges, making them especially suitable for use in social networks, recommendation systems, and fraud detection. We will demonstrate how graph databases help process and analyze complex relational data [39].

Cluster Computing and Distributed Systems

Apache Hadoop Hadoop [40] is a distributed storage and computing framework that efficiently processes large-scale data through the MapReduce model, suitable for batch processing tasks. We will explain how to use Hadoop to handle unstructured data and demonstrate its applications in e-commerce and finance.

Apache Spark Spark [41] offers a faster in-memory computing framework than Hadoop, supporting real-time data stream processing and complex data analysis tasks. This book will show how to leverage SparkâĂŹs advantages in real-time data analysis, machine learning, and large-scale data processing.

0.1.5 Future Competitiveness: Essential Skills in Deep Learning and Data Management

With the widespread adoption of AI technologies, deep learning, and big data management are no longer tools exclusively for data scientists but have become core skills required for professionals across industries. Whether you are a software developer, enterprise manager, or researcher, mastering these technologies will not only help you face future challenges but will also add irreplaceable value to your work [42].

Numerous industries have already been dramatically transformed by the rapid development of AI technologies. Deep learning and big data management skills are becoming increasingly important for technical roles. Deep learning is necessary for developing AI-driven applications, automating business processes, and optimizing system performance. Through big data and artificial intelligence, enterprise managers can optimize decision-making processes to improve efficiency and drive digital transformation [5, 43].

However, the risk of not mastering these skills is evident. As automation systems and intelligent decision-making systems become more prevalent, many traditional roles will be replaced. Only those who can fully utilize deep learning and big data technologies will remain competitive in the evolving job market.

Through learning this book, readers will gain the key technical skills in these two critical fields and develop the ability to apply them practically, positioning themselves favorably for future career success.

0.1.6 Goals of This Book

The goal of this book is to simplify complex technologies, provide intuitive visualizations, and offer real-world case studies to help readers fully grasp the core concepts of deep learning and machine learning while gaining a deep understanding of the key technologies in big data management and analysis. Whether readers are beginners, technical experts, or business managers, this book will offer systematic content to benefit everyone at various levels.

Simplifying Complex Technologies to Lower the Learning Curve

Deep learning and machine learning typically involve complex mathematical derivations and algorithmic implementations, such as linear algebra, calculus, and probability theory. This book aims to simplify these complex topics, helping readers understand their core principles. By using simplified mathematical formulas, intuitive explanations, and accessible analysis, readers will be able to easily grasp key concepts without being overwhelmed by technical details.

Comprehensive Coverage of Applications and Classic Models

To help readers fully understand the practical applications of deep learning and machine learning, this book will cover a wide range of scenarios, from natural language processing (NLP) to computer vision, reinforcement learning, and generative adversarial networks (GANs) [27]. We will provide detailed explanations of classic models such as Transformer, ChatGPT [44], AlphaGo Zero [20], BERT [22], ResNet [21], and YOLO [23], and show how these models work in practice, covering real-world applications in industries such as finance, healthcare, autonomous driving, and security.

These models will be presented in order of their impact on the industry, helping readers prioritize the most important technologies. Through rich case studies and real-world application examples, readers will gain a clear understanding of how these technologies are transforming workflows and decision-making processes in various industries.

Core Data Management Technologies and Tools

The efficient operation of these models is dependent on modern data management technologies, which go beyond deep learning and machine learning. This book will explain in detail how to use SQL and NoSQL databases to store and manage large-scale data, how to handle complex relational data with graph databases (such as Neo4j), and how to process vast amounts of data using cluster computing frameworks (such as Hadoop [40] and Spark [41]).

Through real-world case studies, we will demonstrate how to integrate these data management technologies with deep learning to build scalable solutions, helping readers master the entire process of big data analysis, from data collection, storage, and cleaning to modeling and prediction.

Visualization and Practice-Driven Learning

By using visualization tools throughout the book, readers will gain a better understanding of how neural network layers function, how CNNs process images, and how model parameters are optimized using gradient descent. This complex algorithm can be intuitively understood by using diagrams and data visualizations.

In addition, this book will provide practical projects and code examples to demonstrate how theory can be applied in practice. Whether itâĂŹs training machine learning models or building big data platforms, readers will be guided step by step through the entire process, easily mastering the transition from theory to application. Each key concept will be accompanied by concrete examples, helping readers apply what they have learned in real-world scenarios.

A Comprehensive and Systematic Learning Path

The content of this book spans from foundational theory to cutting-edge technologies, striving for comprehensive coverage. Readers will follow a systematic learning path, starting from the basic theories of machine learning and deep learning, and gradually progressing to big data management, model optimization, and cluster computing. Whether you are a beginner or an experienced professional, you will find content tailored to your needs and gradually master the entire technological ecosystem.

This book provides readers with a comprehensive understanding of the big data management and analysis ecosystem, such as deep learning and machine learning. Whether you are looking to enhance your current skills or prepare for future career growth, this book will provide you with a complete set of knowledge and skills to help you stay competitive in an AI-driven era.

Part I

Handy Appetizer

Chapter 1

Getting Started with Python

1.1 Why Python?

Python is one of the easiest programming languages to learn. ItâĂŹs known for its clean, readable syntaxâĂŤwriting Python is almost like writing English. Whether you're interested in data science, building websites, or creating machine learning models, Python [45] is the perfect starting point.

1.1.1 What is Python Used For?

- **Data Science:** Analyze large datasets with libraries like `pandas` and `numpy`.
- **Web Development:** Build web applications using frameworks like `Flask` and `Django`.
- **Machine Learning:** Create predictive models with libraries such as `scikit-learn` and `TensorFlow`.
- **Automation:** Automate repetitive tasks like renaming files or scraping data from websites.

1.2 What is an IDE?

An Integrated Development Environment (IDE) is a tool that helps you write, run, and test code. Python comes with a built-in IDE called **IDLE**. ItâĂŹs simple, easy to use, and perfect for beginners. While there are other IDEs available, such as VS Code and PyCharm, we will focus on IDLE for now.

In future chapters, weâĂŹll explore other IDEs and tools that might be more suitable for larger projects.

1.3 Setting Up Python

LetâĂŹs get Python up and running on your computer. These instructions are for complete beginners, so follow along step by step.

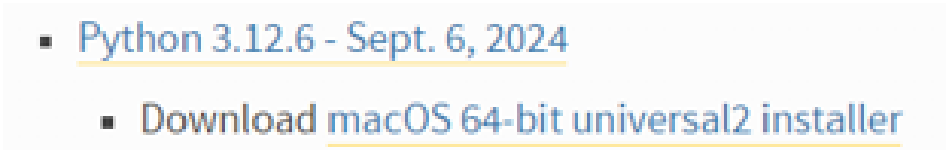
1.3.1 Step 1: Download Python

1. Open your web browser and go to the official Python website: python.org/downloads.

2. Depending on your operating system, choose the appropriate version to download:
   - For **Windows**, click the large button that says **Download Python 3.x.x** (e.g., Python 3.12.6) as shown in the image below.



   - For **MacOS**, follow this link: macOS version. Choose the latest stable release and download the universal installer, as shown in the two images below.



3. Make sure you download the version marked as Python 3.x (e.g., Python 3.11).

1.3.2 Step 2: Install Python

For Windows:

1. Once the download is complete, run the installer by double-clicking on it.
2. Before proceeding, check the box that says **Add Python to PATH**. This is important!
3. Click **Install Now** and wait for the installation to complete. YouâĂŹll see a message that says **Setup was successful**.

For MacOS:

1. Open the downloaded `.pkg` file.
2. Follow the prompts by clicking **Continue** and **Agree**.
3. Once the installation is finished, open **Terminal** and type the following to verify Python is installed:

```
python3 --version
```

For Linux:

1. Many Linux distributions come with Python pre-installed. You can check by opening **Terminal** and typing:

```
python3 --version
```

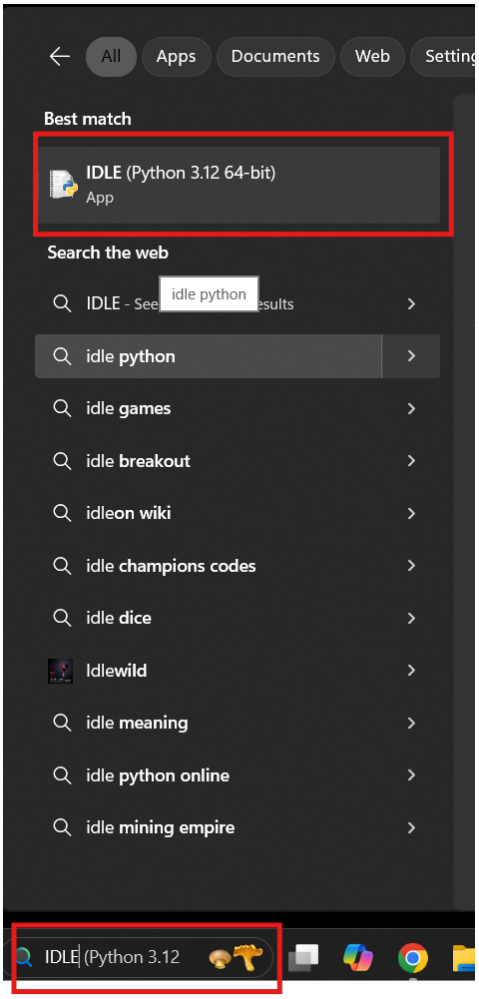
2. If Python is not installed, use the following command:

```
sudo apt-get install python3
```

1.4 Step 3: Using IDLE - Your First Python IDE

Python comes with its own built-in IDE called **IDLE** (Integrated Development and Learning Environment). You donâĂŹt need to install anything extra! Once Python is installed, you can start using IDLE right away to write and run Python code.

- **Windows and Mac:** After installing Python, search for `IDLE` in your start menu (or Finder on Mac) and open it. The image below shows how it looks on a Windows machine:



- **Linux:** Open Terminal and type `idle3`.

In IDLE, you can type Python code and run it instantly. ItâĂŹs a perfect way to get familiar with Python, especially if youâĂŹve never programmed before.

1.5 Your First Python Program

Now that you have Python installed and IDLE open, letâĂŹs write your first Python program! This will be the simplest program you can imagine, but itâĂŹs a great way to see Python in action.

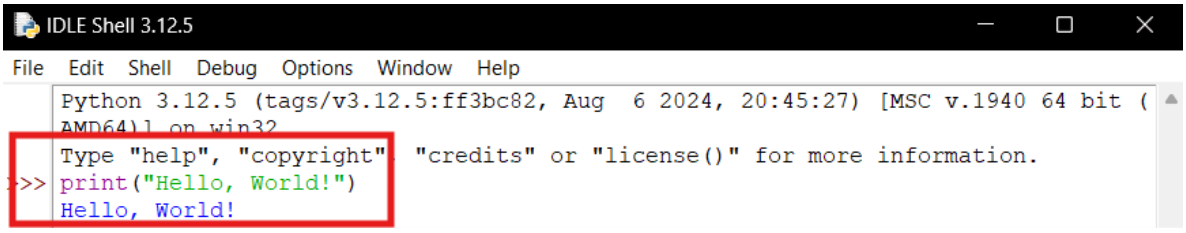
1.5.1 Step-by-Step Instructions

1. Open **IDLE**.
2. In the IDLE window, type the following code:

```
print("Hello, World!")
```

3. Press `Enter`.

You should see the following output:




```
Hello, World!
```

ThatâĂŹs it! YouâĂŹve just written and run your first Python program. This simple code tells Python to print the text `"Hello, World!"` on the screen.

1.6 Quick Recap

- Python is a beginner-friendly language thatâĂŹs easy to learn and powerful to use.
- You installed Python and used its built-in IDE, **IDLE**, to write and run Python code.
- You successfully wrote your first Python program!

In the next chapter, weâĂŹll dive into real-world Python applications. You'll see how Python can be used for data analysis, automation, and more.

Chapter 2

Understanding Python Packages and `pip`

2.1 What are Python Packages?

Python packages are collections of modules (i.e., pre-written Python code) that provide functions and tools for specific tasks. For example, if you need to work with data visualization, you can use the `matplotlib` package. If youâĂŹre doing data analysis, `pandas` can be useful. Packages make coding faster and more efficient by providing reusable code.

2.2 Why Do We Need Packages?

Packages save time by providing pre-built tools, so you donâĂŹt have to write everything from scratch. If you want to plot graphs, instead of writing your own plotting code, you can install a package like `matplotlib` or `seaborn` and use its functions. This way, you focus on solving the problem rather than building the tools from scratch.

2.3 How to Use Packages in Python

To use a package in Python, you first need to install it. Python uses a tool called `pip` to download and install packages from the Python Package Index (PyPI). Once installed, you can import the package into your code.

2.4 What is `pip`?

`pip` is a package manager that allows you to install and manage Python packages. ItâĂŹs included with most Python installations, so you donâĂŹt need to install `pip` separately [46].

2.5 Why Do We Need pip?

pip helps manage all the external libraries you want to use in your Python projects. Whether you need libraries for data analysis, machine learning, or web development, pip makes the installation process easy.

2.6 Example: Installing seaborn Using pip

LetâĂŹs say you want to use the seaborn package, which is a powerful library for creating statistical graphics. YouâĂŹll first need to install it using pip.

```
pip install seaborn
```

The following screenshot shows what the installation process looks like in a command line:



As you can see, pip automatically downloads all the required dependencies, installs them, and finishes with a success message.

2.7 Use seaborn to plot

Seaborn [47] is a powerful and user-friendly Python library built on top of Matplotlib [48], designed specifically for creating informative and attractive statistical graphics. It simplifies the process of creating complex visualizations by providing high-level interfaces for drawing various types of plots, such as categorical plots, distribution plots, and regression plots. Seaborn also integrates well with Pandas, allowing for easy plotting of data directly from DataFrames, making it a convenient tool for data exploration and analysis. Its default styles and color palettes also enhance the aesthetics of the plots without requiring extensive customization.

2.7.1 Example: Using seaborn to Create a Simple Plot

Now that seaborn is installed, letâĂŹs use it to create a simple plot. HereâĂŹs how you can use seaborn inside IDLE to generate a plot:

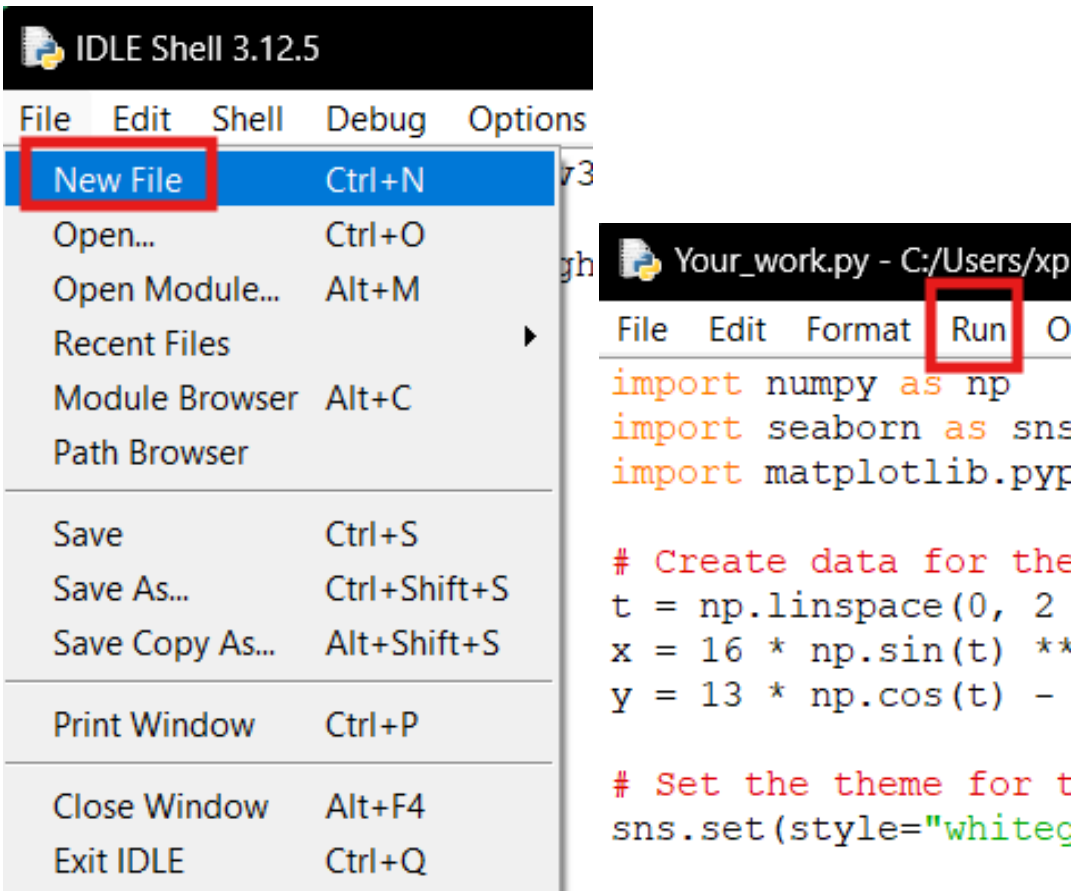
```
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt

# Create data for the heart shape
t = np.linspace(0, 2 * np.pi, 1000)
x = 16 * np.sin(t) ** 3
y = 13 * np.cos(t) - 5 * np.cos(2 * t) - 2 * np.cos(3 * t) - np.cos(4 * t)
```

```
# Set the theme for the plot
sns.set(style="whitegrid")

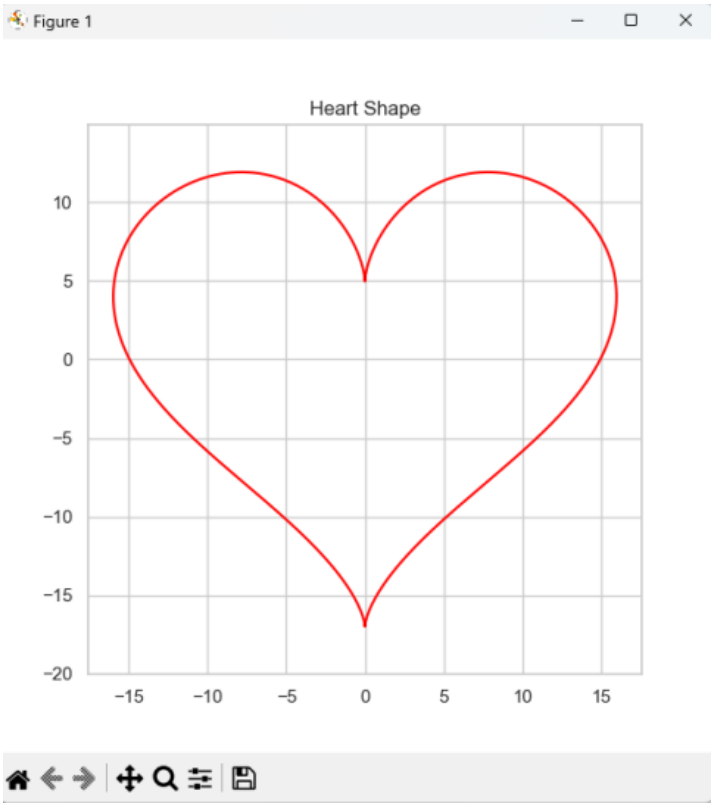
# Create the plot
plt.figure(figsize=(6, 6))
plt.plot(x, y, color="red")
plt.title("Heart Shape")
plt.axis("equal")
plt.show()
```

The image below shows how this looks when running the code in IDLE:



This code uses `seaborn` to load a sample dataset of restaurant tips and create a scatter plot showing the relationship between the total bill and the tip amount.

After running this code, you will see the following plot:



As you can see, the plot visually represents the relationship between the total bill and the tip amount. `seaborn` makes it very easy to create beautiful visualizations with just a few lines of code.

2.7.2 Another Example: Visualizing the `tips` Dataset

The second example demonstrates how to use the `seaborn` package to create a simple scatter plot from the popular "tips" dataset [49], which is included in `seaborn`. This dataset contains information about restaurant tips, such as total bill, tip amount, and various other factors.

HereâĂŹs the code to visualize the relationship between the total bill and the tip amount:

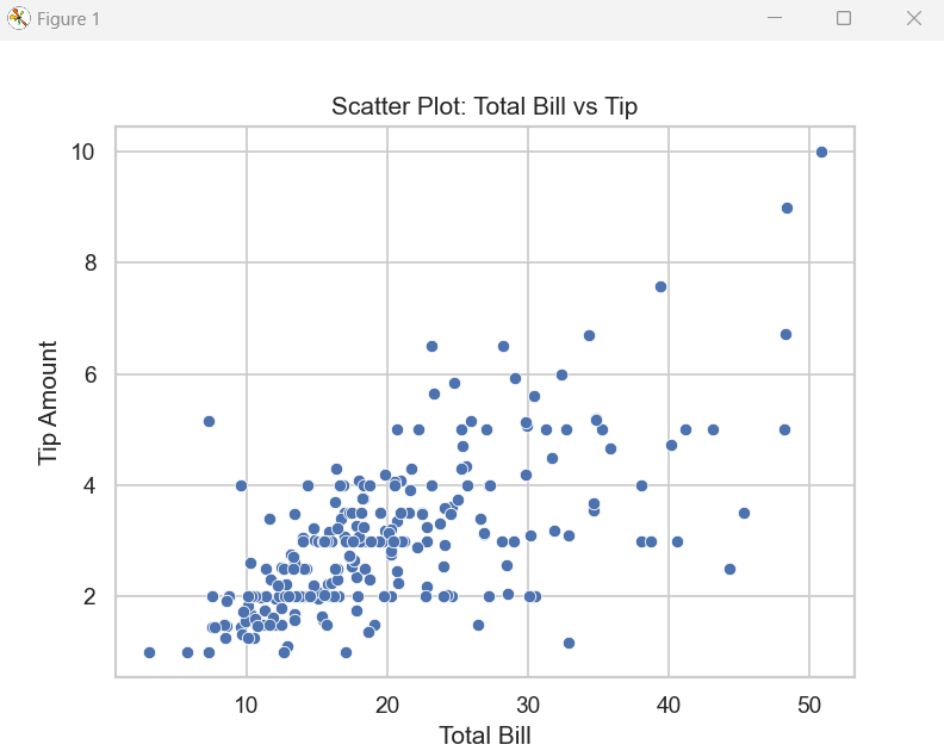
```
import seaborn as sns
import matplotlib.pyplot as plt

# Load the 'tips' dataset
tips = sns.load_dataset("tips")

# Set the theme for the plot
sns.set(style="whitegrid")

# Create a scatter plot
sns.scatterplot(x="total_bill", y="tip", data=tips)

# Show the plot
plt.title("Scatter Plot: Total Bill vs Tip")
plt.xlabel("Total Bill")
plt.ylabel("Tip Amount")
plt.show()
```



user may see output shown as above. Now we have seen the example of using function and dataset in packages, in the next chapter, we will dive into more advanced topics, including machine learning and using Python to build models.

2.8 Summary

- Python packages are collections of code that allow you to perform various tasks efficiently.
- `pip` is the tool used to install and manage Python packages.
- We installed the `seaborn` package using `pip`, and used it to generate a simple scatter plot.

Chapter 3

Machine Learning 101

3.1 What is Machine Learning?

Machine Learning (ML) is a field of computer science that enables computers to learn from data without being explicitly programmed. Think of it as teaching a computer to recognize patterns, make decisions, or predict outcomes based on examples (data) that we give it.

Instead of writing rules for the computer to follow, we provide it with data, and it learns the rules by itself.

3.2 What Can Machine Learning Do?

Machine Learning is everywhere around us. Here are a few examples of what ML can do:

- **Predicting Outcomes:** ML can predict what might happen next. For example, predicting whether an email is spam or not, or predicting whether a customer will buy a product.
- **Image Recognition:** ML models can recognize faces or objects in photos and videos.
- **Personalization:** Online services like Netflix or Spotify use ML to recommend movies or songs based on your preferences.
- **Speech Recognition:** Voice assistants like Siri or Alexa use ML to understand spoken commands.

3.3 Understanding Machine Learning

It is easiest to understand ML by thinking of it as teaching a child. When you teach a child to recognize animals, you don't explain every animal's rules. Instead, you show them many pictures of animals and tell them what each one is. Eventually, they learn to recognize new animals by comparing them to previous examples.

It works similarly to ML. By providing a computer with a lot of data (examples), we allow it to find patterns on its own. As soon as it learns those patterns, it can make predictions based on new, unseen data.

3.4 Introducing `scikit-learn`

`scikit-learn` is one of the most popular and beginner-friendly packages for machine learning in Python [50]. It provides easy-to-use tools for building machine learning models, like decision trees, linear regression, and more.

3.4.1 Installing `scikit-learn`

To install `scikit-learn`, use `pip` as we did in the previous chapter. Open your terminal and run:

```
pip install scikit-learn
```

Once installed, you can import it and use it in your Python code to create machine learning models.

3.5 Training a Linear Regression Model on the Diabetes Dataset

Now, letâĂŹs use `scikit-learn` to train a **Linear Regression** model on the Diabetes dataset[51].

What is Linear Regression? Linear Regression is one of the simplest machine learning models. ItâĂŹs used to predict a numerical value (a continuous variable). For example, predicting a personâĂŹs weight based on their height. The idea is to find the best-fitting line that represents the relationship between input features (e.g., height) and the output (e.g., weight).

In this case, we will predict a diabetes progression value based on certain medical features.

3.5.1 Step 1: Load the Diabetes Dataset and Train a Linear Regression Model

HereâĂŹs the Python code for trainning a Linear Regression model on the Diabetes dataset using `scikit-learn`. We can load the dataset directly from /textttscikit-learn.

```
# Import necessary libraries
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_diabetes
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error

# Step 1: Load the Diabetes dataset
diabetes = load_diabetes()
X = diabetes.data[:, np.newaxis, 2] # Use only one feature (BMI) for simplicity
y = diabetes.target # The target variable (progression of diabetes)

# Step 2: Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Step 3: Create and train the Linear Regression model
lr = LinearRegression()
lr.fit(X_train, y_train)

# Step 4: Make predictions on the test set
```

```
y_pred = lr.predict(X_test)

# Step 5: Calculate the Mean Squared Error (MSE)
mse = mean_squared_error(y_test, y_pred)
print(f"Mean Squared Error: {mse:.2f}")

# Step 6: Visualize the scatter plot and regression line
plt.figure(figsize=(8, 6))

# Scatter plot of the training data
plt.scatter(X_train, y_train, color="blue", label="Training data", alpha=0.6)

# Scatter plot of the test data
plt.scatter(X_test, y_test, color="green", label="Test data", alpha=0.6)

# Regression line (use entire dataset for better visualization)
plt.plot(X, lr.predict(X), color="red", linewidth=2, label="Regression line")

# Add labels and title
plt.title("Linear Regression on Diabetes Dataset (BMI vs Progression)")
plt.xlabel("BMI (Body Mass Index)")
plt.ylabel("Diabetes Progression")
plt.legend()

# Show the plot
plt.show()
```

Explanation of the Code:

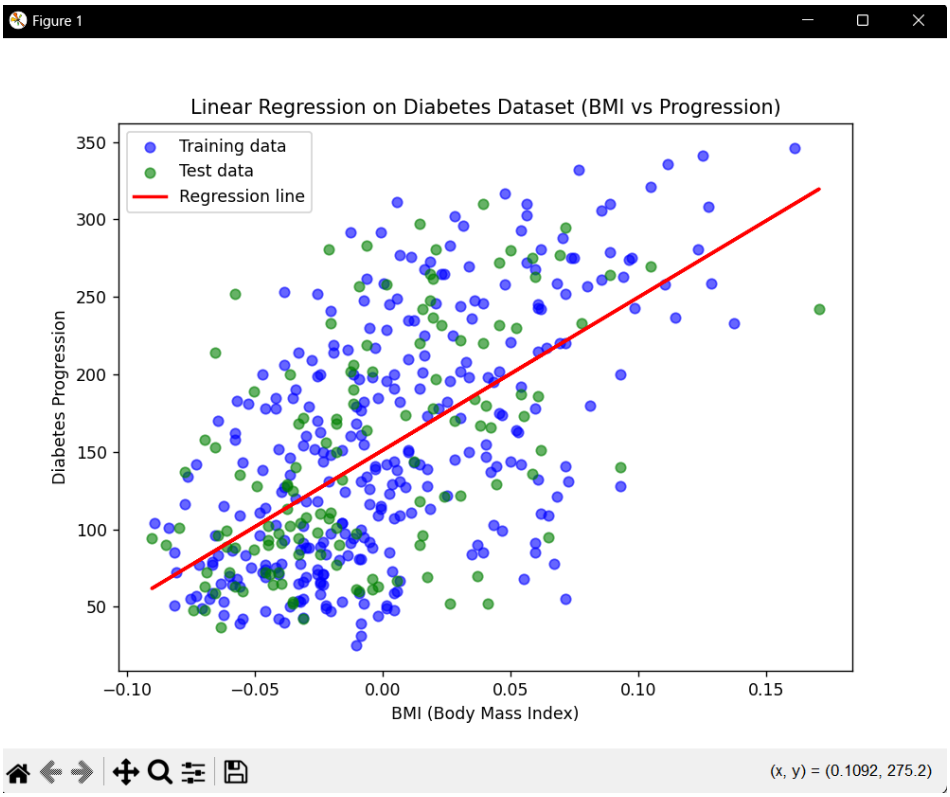
- `X = diabetes.data[:, np.newaxis, 2]`: WeâĂŹre selecting only one feature, BMI (Body Mass Index), to make the regression simpler and easier to visualize. This limits the model to predicting diabetes progression based on BMI alone.
- `plt.scatter()`: Creates scatter plots of the training data (in blue) and test data (in green) to show the relationship between BMI and diabetes progression.
- `plt.plot()`: Draws the regression line (in red) that represents the linear relationship between BMI and diabetes progression.
- The plot includes clear labels, colors, and a legend to distinguish between training data, test data, and the regression line.

3.6 Visualizing the Result

This plot visually shows how well the linear regression model fits the data. The blue and green points represent the actual data (BMI vs diabetes progression), and the red line is the linear regression line predicted by the model. The closer the points are to the red line, the better the model's predictions are.

- **Blue points:** Training data used to fit the model.

- **Green points:** Test data, which the model hasnâĂŹt seen before.
- **Red line:** The linear regression line representing the predicted relationship between BMI and diabetes progression.



3.7 Training a Logistic Regression Model on the Breast Cancer Dataset

LetâĂŹs use scikit-learn to train a Logistic Regression model on the Breast Cancer dataset.

What is Logistic Regression? Logistic Regression is a widely used machine learning algorithm for binary classification tasks. It models the probability that a given input belongs to one of two classes by fitting the data to a logistic function (also known as the sigmoid function). Unlike Linear Regression, which predicts continuous values, Logistic Regression is used to classify inputs into categories, such as "yes" or "no" (binary classification).

In this case, we will use the Breast Cancer dataset from scikit-learn to predict whether a tumor is benign or malignant based on specific medical features.

3.7.1 Load the Breast Cancer Dataset and Train a Logistic Regression Model

HereâĂŹs the Python code to train a Logistic Regression model on the Breast Cancer dataset, see **Figure 3.1** for our final result:

```
# Import necessary libraries
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
```

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, confusion_matrix, roc_curve, auc

# Step 1: Load the Breast Cancer dataset
cancer = load_breast_cancer()
X = cancer.data # Using all features for this model
y = cancer.target # Binary target (1: malignant, 0: benign)

# Step 2: Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Step 3: Create and train the Logistic Regression model
lr = LogisticRegression(max_iter=10000)
lr.fit(X_train, y_train)

# Step 4: Make predictions on the test set
y_pred = lr.predict(X_test)

# Step 5: Calculate the accuracy of the model
accuracy = accuracy_score(y_test, y_pred)

# Step 6: Visualize the training and testing data with the decision boundary
x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.01),
                     np.arange(y_min, y_max, 0.01))

plt.figure(figsize=(8, 6))
plt.contourf(xx, yy, Z, alpha=0.3, cmap=ListedColormap(('orange', 'blue')))

# Scatter plot for training data
plt.scatter(X_train[:, 0], X_train[:, 1], c=y_train, s=50, edgecolor='k', marker='o',
            cmap=ListedColormap(('red', 'green')), label='Training Data')

# Scatter plot for testing data
plt.scatter(X_test[:, 0], X_test[:, 1], c=y_test, s=50, marker='s', edgecolor='k',
            cmap=ListedColormap(('red', 'green')), label='Testing Data')

# Labels and title
plt.title('Logistic Regression Decision Boundary\nwith Training and Testing Data')
plt.xlabel(cancer.feature_names[0])
plt.ylabel(cancer.feature_names[1])
plt.legend()
plt.show()
```

Explanation of the Code:

- `X = cancer.data`: We are using all features available in the dataset to predict the outcome (malignant or benign).
- `y = cancer.target`: This represents the binary classification labels (1 for malignant tumors, 0

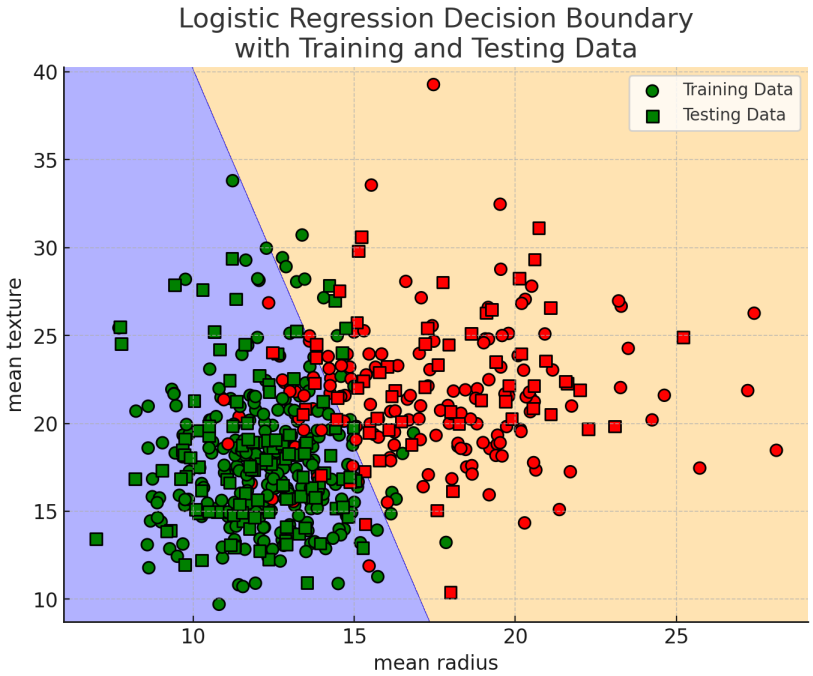


Figure 3.1: The graph displays the decision boundary of a Logistic Regression model trained on two features from the Breast Cancer dataset. The circles represent the training data, while the squares represent the testing data. Colors indicate the class labels: red for malignant tumors and green for benign tumors. The decision boundary separates the two classes, with the orange and blue regions indicating the predicted classification zones for malignant and benign cases, respectively. The plot effectively visualizes how well the model differentiates between the two tumor types.

for benign).

- `LogisticRegression(max_iter=10000)`: Creates a Logistic Regression model. The parameter `max_iter=10000` ensures that the optimization converges, as the default value may be insufficient for some datasets.
- `predict_proba()`: Returns the probability estimates for the positive class (malignant tumors). This is useful for generating the ROC curve.

3.8 Using ChatGPT to Generate Machine Learning Code

ChatGPT is an AI language model developed by OpenAI [44]. You can use it to assist with generating code, understanding complex topics, and even debugging issues. If you're unsure how to write certain Python code, like training a Linear Regression model, ChatGPT can help by generating the code for you.

To use ChatGPT, go to chat.openai.com, log in (or sign up if you're a new user), and then start a conversation by typing your request in plain language.

3.8.1 How to Prompt ChatGPT for Code

When you interact with ChatGPT, the more specific you are in your prompt, the more accurate and helpful the response will be. To generate a Linear Regression training code, you can provide the following instructions:

Example Prompt:

`Can you generate Python code using scikit-learn to train a Linear Regression model on the Diabetes dataset? Please include code to split the data into training and testing sets, train the model, make predictions, and calculate the Mean Squared Error. Add in-line comments to explain each step.`

3.8.2 Example Response from ChatGPT

HereâĂŹs an example of the code ChatGPT might generate for this prompt:

```python
# Import necessary libraries
from sklearn.datasets import load_diabetes
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error

# Load the Diabetes dataset
diabetes = load_diabetes()
X = diabetes.data # Features
y = diabetes.target # Labels

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Create the Linear Regression model
lr = LinearRegression()

# Train the model
lr.fit(X_train, y_train)

# Make predictions on the test set
y_pred = lr.predict(X_test)

# Calculate and print the Mean Squared Error
mse = mean_squared_error(y_test, y_pred)
print(f"Mean Squared Error: {mse:.2f}")
```

3.9 Summary

- Machine Learning enables computers to learn from data and make predictions.
- `scikit-learn` is a popular Python package for building ML models.

- We used `scikit-learn` to train a Linear Regression [52] model on the Diabetes dataset and evaluated it using Mean Squared Error.
- Linear Regression is a simple model for predicting continuous values based on input features.
- ChatGPT can be a valuable tool for generating code and understanding machine learning concepts.

Chapter 4

Deep Learning 101

4.1 What is Deep Learning?

Deep Learning (DL) is a type of Machine Learning that focuses on models called **neural networks**. These models are inspired by how the human brain works and are very powerful at recognizing patterns in large amounts of data.

In Deep Learning, neural networks have several layers (which is why it's called "deep"). Each layer learns to represent data in a more complex way. For example, if you give it images, early layers may recognize simple shapes like edges, while deeper layers recognize whole objects like numbers or faces.

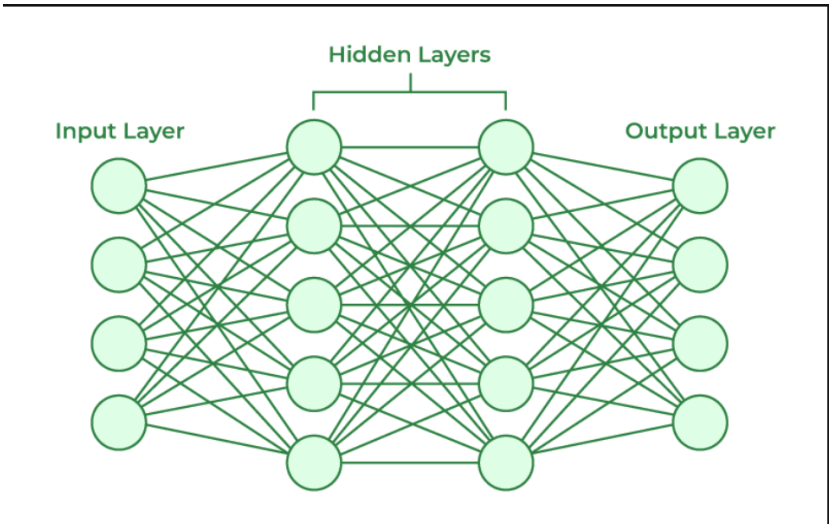
4.2 What Can Deep Learning Do?

Deep Learning can solve very complex tasks that traditional algorithms struggle with. Some examples include:

- **Image Recognition:** Recognizing objects in pictures, like identifying handwritten digits or faces.
- **Language Understanding:** Translating languages, understanding commands, or generating text (e.g., ChatGPT and Gemini).
- **Speech Recognition:** Turning spoken words into text or understanding voice commands, like in Siri or Alexa.
- **Game Playing:** Deep Learning models can learn to play and master complex games like Chess or Go.

4.3 How Does Deep Learning Work?

The key idea in Deep Learning is the **neural network**. A neural network is made up of layers of connected nodes, called **neurons**. Each connection between neurons has a weight, which changes as the network learns from data.



In simple terms:

- **Input Layer:** Receives the raw data (e.g., images or text).
- **Hidden Layers:** These layers process the data, learning patterns from it.
- **Output Layer:** Produces the final prediction, like classifying an image as a digit.

The more layers a network has, the more complex patterns it can learn. However, deep networks also need lots of data and computing power to work well.

4.4 Mainstream Deep Learning Frameworks

Deep learning involves many complex mathematical algorithms, which makes it difficult to build models from scratch. Fortunately, there are now several mainstream frameworks that allow us to bypass the mathematical details and write simple code to implement deep learning models. Two of the most popular frameworks are **TensorFlow 2** [53] and **PyTorch** [54].

Below is a table comparing the pros and cons of each framework:

TensorFlow 2	PyTorch
Pros	
• Simplified API, making it more beginner-friendly compared to previous versions. • Strong support for deployment and production environments. • Well-documented with a large and active community. • Developed by Google, it updates quickly. • It was once very popular and widely used.	• Easier to understand and use, especially for researchers and those who want flexibility. • Dynamic computation graph allows for real-time tracking of the model's workflow and results. • Increasingly popular in both academia and industry. • It is the most used architecture currently and is still growing. Very promising.
Cons	
• Despite improvements, still has a steeper learning curve compared to PyTorch. • Debugging can sometimes be less intuitive due to its static computation graph. • Updates are too fast and always use different APIs, which can cause serious incompatibility issues even if they are all TensorFlow 2.x. • There is a lot of TensorFlow 1.x code on the market, but the incompatibility between TensorFlow 1.x and TensorFlow 2.x makes it difficult to migrate the code. • The number of people using this architecture is decreasing.	• Historically less mature for large-scale production and deployment compared to TensorFlow. • Slightly smaller ecosystem for deployment tools. • The code encapsulation level is low, Not very beginner friendly.
*** It is strongly recommended that beginners focus their learning on PyTorch**	

Table 4.1: Comparison of TensorFlow 2 and PyTorch

TensorFlow 2 and PyTorch APIs are quite similar, but switching between frameworks is still not a trivial task. However, with the advent of AI tools like ChatGPT, switching frameworks has become much easier. To provide readers with a clear comparison between TensorFlow 2 and PyTorch, we will present an example implemented in both frameworks below, allowing you to choose the framework that best suits your needs.

4.5 Introducing `tensorflow` (TensorFlow 2)

`tensorflow 2`, commonly known as TensorFlow, is a widely used tool for building and training deep learning models. It's scalable, beginner-friendly, and widely adopted in industry.

4.5.1 Installing `tensorflow`

To get started with `tensorflow`, you can install it using `pip`:

```
# Install TensorFlow
pip install tensorflow
```

4.6 Training a Neural Network on the MNIST Dataset

The **MNIST dataset** [55] consists of 70,000 images of handwritten digits (0 to 9). We will use it to train a simple neural network that can recognize these digits.

4.6.1 Step-by-Step: Training a Neural Network with TensorFlow 2

Below is the equivalent Python code for training a neural network on the MNIST dataset using TensorFlow 2:

```
# Import necessary libraries
import tensorflow as tf
from tensorflow.keras import layers, models
from tensorflow.keras.datasets import mnist

# Step 1: Load and preprocess the MNIST dataset
(x_train, y_train), (x_test, y_test) = mnist.load_data()

# Normalize the images to values between 0 and 1
x_train, x_test = x_train / 255.0, x_test / 255.0

# Step 2: Define a simple neural network model
model = models.Sequential([
    layers.Flatten(input_shape=(28, 28)), # Flatten the input image (28x28) to a 1D array
    layers.Dense(128, activation='relu'), # Hidden layer with 128 neurons and ReLU activation
    layers.Dense(64, activation='relu'), # Another hidden layer with 64 neurons
    layers.Dense(10) # Output layer with 10 neurons (one for each digit)
])

# Step 3: Compile the model
model.compile(optimizer='adam', # Adam optimizer
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])

# Step 4: Train the neural network
model.fit(x_train, y_train, epochs=5, batch_size=64)

# Step 5: Test the neural network
test_loss, test_acc = model.evaluate(x_test, y_test, verbose=2)
print(f"Accuracy: {test_acc * 100:.2f}%")
```

Explanation of the Code:

- `mnist.load_data()`: Loads the MNIST dataset.
- `x_train / 255.0`: Normalizes the pixel values between 0 and 1.
- `Sequential()`: Defines a sequential model where layers are added one by one.
- `Flatten()`: Converts the 28x28 images into a flat vector.
- `Dense()`: A fully connected layer. The last layer has 10 neurons for the 10 digit classes.
- `compile()`: Specifies the optimizer, loss function, and evaluation metrics.
- `fit()`: Trains the model on the training data for a specified number of epochs.
- `evaluate()`: Tests the model and returns the loss and accuracy.

4.7 Training Output and Accuracy

During the training process, you will see continuous updates on the model's performance after each batch of data. These updates include metrics such as the loss and accuracy. Below is an example of how this output looks during training, where the accuracy at the end of training is highlighted:



As shown in the screenshot, the final accuracy of the model is 97.76%. This accuracy value represents how well the model is performing on the test dataset after training. The lower the loss and the higher the accuracy, the better the model is at making correct predictions.

4.8 Introducing `torch` (PyTorch)

`torch`, also known as PyTorch, is another popular tool for building and training deep learning models. It's flexible, easy to use, and widely adopted by both researchers and developers.

4.8.1 Installing `torch`

To get started with `torch`, we need to install it. HereâĂŹs how you can install PyTorch using `pip`:

```
# Install PyTorch with CPU support (no GPU needed)
pip install torch torchvision
```

`torchvision` is an extra package that helps with handling image data, like the MNIST dataset weâĂŹll use in the next section.

4.9 Training a Neural Network on the MNIST Dataset

The **MNIST dataset** is a collection of 70,000 images of handwritten digits (0 to 9). WeâĂŹll use it to train a simple neural network that can recognize these digits.

4.9.1 Step-by-Step: Training a Neural Network with PyTorch

HereâĂŹs a Python example of training a neural network using the MNIST dataset with PyTorch. We will load the dataset, define the network, train it, and evaluate its accuracy.

```
# Import necessary libraries
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import DataLoader
from torchvision import datasets, transforms

# Step 1: Prepare the MNIST dataset
transform = transforms.Compose([transforms.ToTensor(), transforms.Normalize((0.5,), (0.5,))])
train_data = datasets.MNIST(root='mnist_data', train=True, transform=transform, download=True)
test_data = datasets.MNIST(root='mnist_data', train=False, transform=transform, download=True)

# Step 2: Create data loaders for training and testing
train_loader = DataLoader(train_data, batch_size=64, shuffle=True)
test_loader = DataLoader(test_data, batch_size=64, shuffle=False)

# Step 3: Define a simple neural network
class NeuralNet(nn.Module):
    def __init__(self):
        super(NeuralNet, self).__init__()
        self.fc1 = nn.Linear(28 * 28, 128) # Input layer (28x28 image = 784)
        self.fc2 = nn.Linear(128, 64) # Hidden layer
        self.fc3 = nn.Linear(64, 10) # Output layer (10 digits)

    def forward(self, x):
        x = x.view(-1, 28 * 28) # Flatten the image
        x = torch.relu(self.fc1(x)) # Apply ReLU activation to input
        x = torch.relu(self.fc2(x)) # Apply ReLU to hidden layer
        x = self.fc3(x) # Output the result (logits)
        return x
```

```
31
32 # Step 4: Initialize the model, loss function, and optimizer
33 model = NeuralNet()
34 criterion = nn.CrossEntropyLoss() # Loss function for classification
35 optimizer = optim.Adam(model.parameters(), lr=0.001) # Optimizer to update weights
36
37 # Step 5: Train the neural network
38 epochs = 5
39 for epoch in range(epochs):
40     for images, labels in train_loader:
41         optimizer.zero_grad() # Clear gradients from previous step
42         output = model(images) # Forward pass (make prediction)
43         loss = criterion(output, labels) # Compute loss
44         loss.backward() # Backpropagation
45         optimizer.step() # Update weights
46
47     print(f"Epoch {epoch+1}/{epochs}, Loss: {loss.item()}")
48
49 # Step 6: Test the neural network
50 correct = 0
51 total = 0
52 with torch.no_grad(): # No need to compute gradients during testing
53     for images, labels in test_loader:
54         output = model(images)
55         _, predicted = torch.max(output.data, 1) # Get predicted digit
56         total += labels.size(0)
57         correct += (predicted == labels).sum().item()
58
59 accuracy = 100 * correct / total
60 print(f"Accuracy: {accuracy:.2f}%")
```

Explanation of the Code:

- `transforms.Compose()`: Normalizes and converts images to tensors.
- `DataLoader()`: Loads the dataset in batches for faster training.
- `NeuralNet()`: Defines a basic neural network with one hidden layer.
- `optimizer.zero_grad()`: Resets gradients before updating weights.
- `loss.backward()`: Computes the gradient for backpropagation.
- `torch.max()`: Finds the highest probability class (digit) for each prediction.

4.10 Understanding Neural Networks (Simplified)

Neural networks are made of layers of neurons, and each connection between neurons has a weight that the network adjusts during training. HereâĂŹs a simple breakdown of how it works:

- **Input Layer:** Receives raw data (in this case, an image of a digit).

- **Hidden Layer:** Processes the input and looks for patterns (e.g., lines, shapes).
- **Output Layer:** Produces the final result (the digit the network thinks it is).

The network learns by adjusting its weights based on the errors it makes during training. Over time, it gets better at making accurate predictions.

4.11 Using ChatGPT to Generate Deep Learning Code

If you're unsure how to write deep learning code or need help, ChatGPT can be a helpful tool. You can ask it to generate code for training a neural network or to explain a concept in detail.

4.11.1 How to Prompt ChatGPT for Code

When asking ChatGPT for deep learning code, it's important to be clear in your request. HereâĂŹs an example prompt you could use:

TensorFlow 2 Version

Example Prompt:

```
Can you generate Python code using TensorFlow 2 to train a neural network on the MNIST dataset?
Please include code to load the dataset, define a simple neural network, train it, and
evaluate its accuracy. Add in-line comments to explain each step.
```

Example Response from ChatGPT

HereâĂŹs an example of what ChatGPT might generate in TensorFlow 2:

```python
# Import necessary libraries
import tensorflow as tf
from tensorflow.keras import layers, models
from tensorflow.keras.datasets import mnist
import numpy as np

# Load MNIST dataset
(x_train, y_train), (x_test, y_test) = mnist.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0

# Define the neural network
model = models.Sequential([
    layers.Flatten(input_shape=(28, 28)),
    layers.Dense(128, activation='relu'),
    layers.Dense(64, activation='relu'),
    layers.Dense(10)
])

# Compile the model
model.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])
```

```
# Train the model
model.fit(x_train, y_train, epochs=5, batch_size=64)

# Evaluate the model
test_loss, test_acc = model.evaluate(x_test, y_test, verbose=2)
print(f'Accuracy: {test_acc:.2f}%')
```

PyTorch Version

Example Prompt:

Can you generate Python code using PyTorch to train a neural network on the MNIST dataset? Please include code to load the dataset, define a simple neural network, train it, and evaluate its accuracy. Add in-line comments to explain each step.

Example Response from ChatGPT

HereâĂŹs an example of what ChatGPT might generate:

```
# Import necessary libraries
import torch
import torch.nn as nn
from torchvision import datasets, transforms
from torch.utils.data import DataLoader

# Load MNIST dataset
transform = transforms.Compose([transforms.ToTensor(), transforms.Normalize((0.5,), (0.5,))])
train_data = datasets.MNIST(root='mnist_data', train=True, transform=transform, download=True)
test_data = datasets.MNIST(root='mnist_data', train=False, transform=transform, download=True)

# Create data loaders
train_loader = DataLoader(train_data, batch_size=64, shuffle=True)
test_loader = DataLoader(test_data, batch_size=64, shuffle=False)

# Define the neural network
class NeuralNet(nn.Module):
    def __init__(self):
        super(NeuralNet, self).__init__()
        self.fc1 = nn.Linear(28*28, 128)
        self.fc2 = nn.Linear(128, 64)
        self.fc3 = nn.Linear(64, 10)

    def forward(self, x):
        x = x.view(-1, 28*28)
        x = torch.relu(self.fc1(x))
        x = torch.relu(self.fc2(x))
        x = self.fc3(x)
        return x

# Initialize model, loss function, and optimizer
```

```
model = NeuralNet()
criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model.parameters(), lr=0.001)

# Train the model
for epoch in range(5):
    for images, labels in train_loader:
        optimizer.zero_grad()
        output = model(images)
        loss = criterion(output, labels)
        loss.backward()
        optimizer.step()

# Evaluate the model
correct = 0
total = 0
with torch.no_grad():
    for images, labels in test_loader:
        output = model(images)
        _, predicted = torch.max(output.data, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()

accuracy = 100 * correct / total
print(f'Accuracy: {accuracy:.2f}%')
```

4.12 Summary

- Deep Learning is a type of Machine Learning that uses neural networks to learn from large amounts of data.
- We used PyTorch to train a neural network on the MNIST dataset.
- Neural networks consist of layers that learn patterns in data, adjusting their weights to improve accuracy.
- ChatGPT can assist with generating Deep Learning code and explaining concepts step-by-step.

Chapter 5

Cool AI Applications That Are Changing the Game

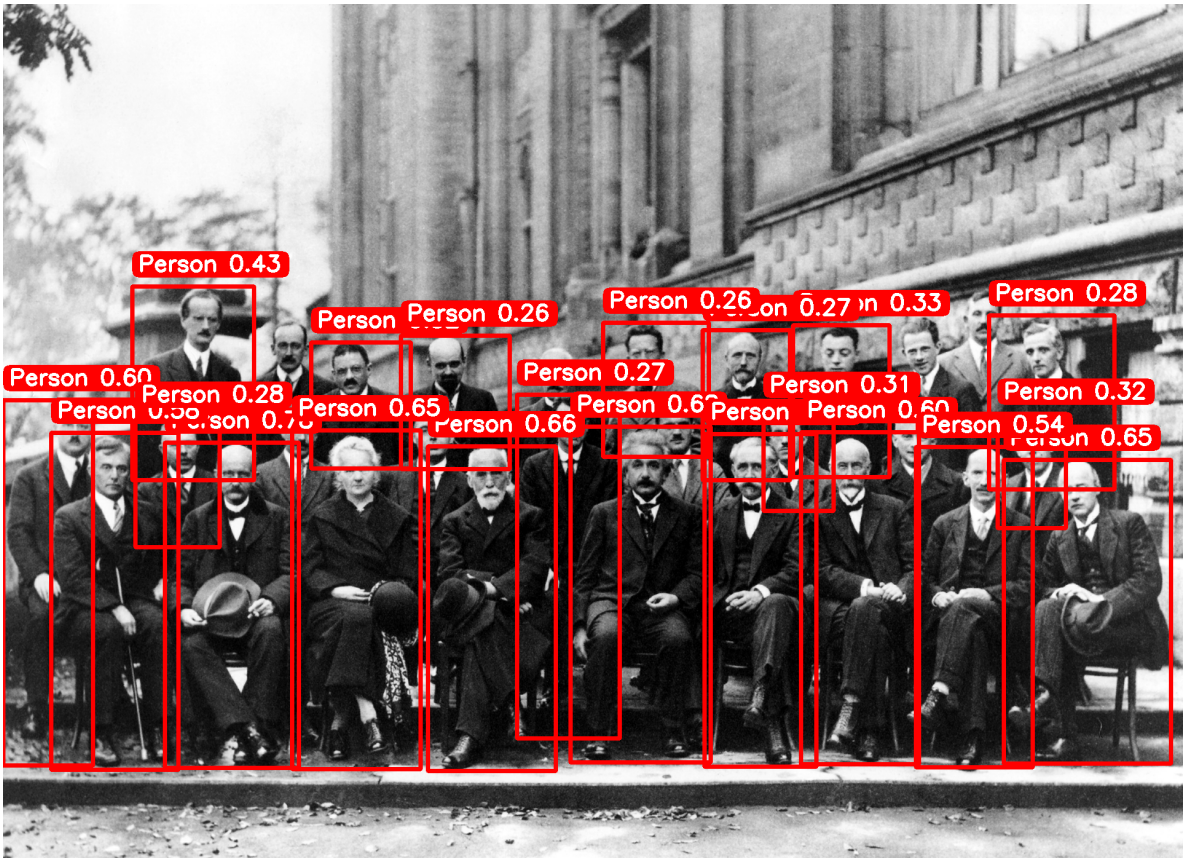


Figure 5.1: Real-world applications of AI object detection (by YOLO v8 [56]). Notice how multiple objects are detected, segmented, and classified in real time, showcasing the efficiency of modern CV algorithms.

AI isnâĂŹt just hype anymoreâĂŤitâĂŹs everywhere, making things that seemed impossible just a few years ago look effortless today. From generating code to identifying objects in images and understanding human language, AI has become the backbone of modern technology. This chapter dives

into some of the coolest AI applications that are not just impressive but are transforming industries and daily life in big ways.

5.1 Cutting-Edge Computer Vision (CV) Tools

5.1.1 Segment Anything Model (SAM)

Ever wish you could instantly cut out an object from a picture or video, no matter how complex the background? **Segment Anything Model (SAM)** [57] is making that wish come true. It lets you segment objects from images and videos in one clickâĂŤperfect for designers, media editors, or even autonomous cars. The SAM web-based GUI is intuitive, allowing you to select and isolate objects with minimal effort. With a simple drag and click, objects are segmented, which makes the editing process more efficient and reduces the need for manual pixel-level adjustments.



Figure 5.2: Left: Original image before SAM processing; Right: SAM's segmented objects. SAM can handle complex scenes, segmenting objects from different categories with precision, making it a revolutionary tool in image processing.

What makes SAM so fascinating: SAM is powered by convolutional neural networks (CNNs) [8], which have been trained on massive datasets to recognize all kinds of objects, no matter the shape, size, or environment. It can handle everything from cutting out a person in a crowded room to isolating a product from a shelf. The web interface provides real-time previews, so users can see the results instantly, making workflows faster.

Inspiration behind SAM: SAM reflects the future of AI popularization âĂŤ AI that requires minimal human input to achieve complex tasks. Developers behind SAM have made it clear that their goal is to make image editing as simple as possible, taking away hours of tedious manual work.

5.1.2 Digital Avatars

From video games to social media, AI-generated **Digital Avatars** [58] are here, and they look more lifelike than ever. These avatars can express emotions, make gestures, and even speak, making them an integral part of virtual reality worlds, online influencers, and more. The user interface of these avatar-creating tools is often straightforward, letting users craft highly realistic avatars with just a few clicks. Whether youâĂŹre a gamer or a content creator, Digital Avatars allow for personal customization that mirrors real-world behaviors and expressions.

Why Digital Avatars are a big deal: Digital avatars, powered by Generative Adversarial Networks (GANs) [27], have made it possible for virtual characters to behave just like humans. In the metaverse, digital avatars represent you, interact with others, and even attend meetings on your behalf. The ease of use and the ability to fine-tune detailsâĂŤsuch as facial expressions and micro-interactionsâĂŤmean anyone can create a digital version of themselves or a fictional character with little technical expertise.

Virtual avatars and influence: Trevor McFedries, founder of Brud, the company behind the AI influencer Lil Miquela, once said, "The future is a world where people will be engaging with personalities that arenâĂŹt even human." That future is now, and digital avatars are leading the charge.

5.2 AI in Natural Language Processing (NLP)

5.2.1 ChatGPT

ChatGPT [44] hardly needs an introduction. This conversational AI from OpenAI can chat with you about anythingâĂŤfrom writing a poem to answering tough technical questions. ChatGPT has become the go-to tool for creative writing, coding, and even providing instant customer service. The web interface is user-friendly, allowing for real-time interaction where the AI responds within seconds, making it highly efficient for both individual users and businesses.

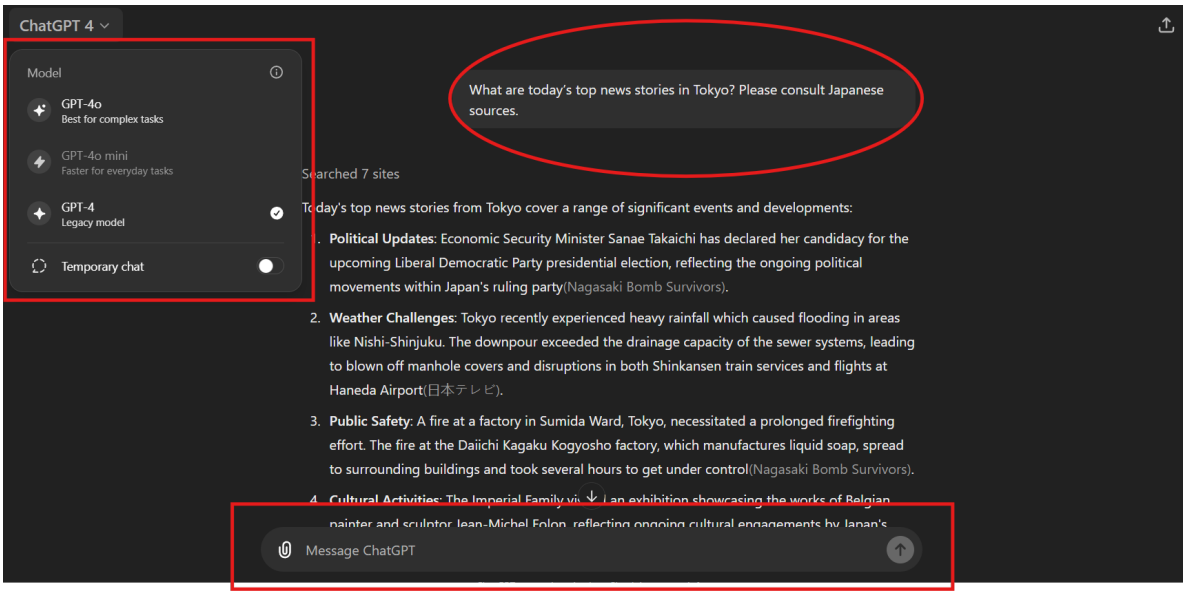


Figure 5.3: ChatGPT in action answering a query. The interface provides a seamless experience, with responses generated in real time, offering both casual conversation and technical support with ease.

The magic behind ChatGPT: ChatGPT runs on the GPT (Generative Pretrained Transformer) [59] architecture, trained on billions of texts to generate human-like responses. ItâĂŹs incredibly versatile and can adapt its responses to different tones and styles, making it a powerful assistant in almost any field. Whether you need it for professional use or casual questions, the platformâĂŹs easy-to-navigate GUI means anyone can engage with the AI effortlessly.

What Sam Altman said: OpenAIâĂŹs CEO Sam Altman once remarked, "AI is the most important technology that humanity has ever developed." ChatGPT certainly lives up to that claim, enabling users worldwide to harness the power of advanced language models with just a few keystrokes.

5.2.2 Gemini: The AI Personal Assistant

Gemini [60] takes AI personal assistants to the next level. Alexa and Siri help you with basic tasks, but Gemini learns from your habits and anticipates your needs. Whether you're drafting emails, scheduling meetings, or responding to texts, Gemini's intuitive interface makes organizing your life simple.

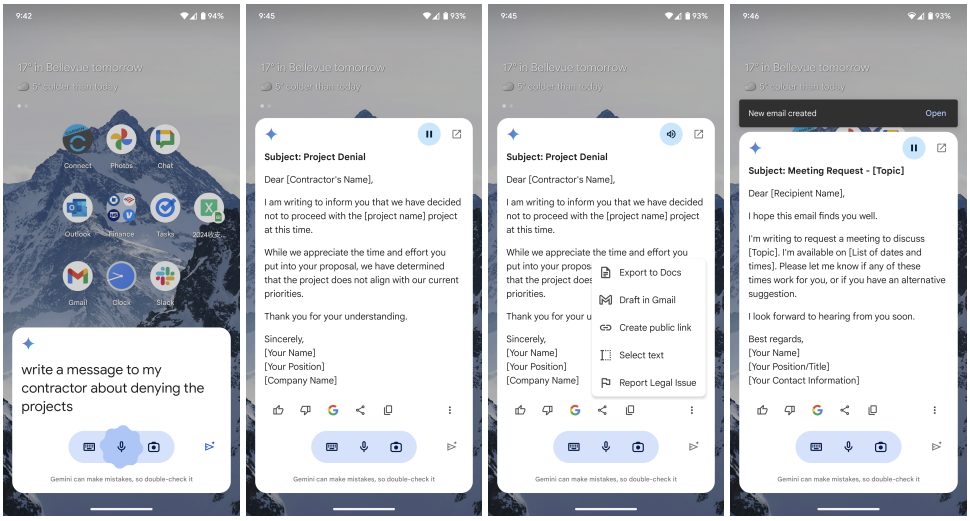


Figure 5.4: Starting a conversation with Gemini. The interface is intuitive and engaging, allowing users to begin tasks with simple commands.Gemini responds efficiently with personalized insights and tasks. Here it shows a list of actions that it can handle without any need for manual setup.Gemini can perform various operations, from scheduling to reminders, shown in this interface as options highlighted for user input.Gemini generating an email draft based on the userâĂŹs preferencesâĂŤwithout typing a single word. This feature shows how the assistant handles repetitive tasks effortlessly.

What sets Gemini apart: Unlike typical assistants, Gemini uses reinforcement learning to get better the more you interact with it. Whether youâĂŹre planning meetings, answering emails, or setting reminders, Gemini can handle all of it without skipping a beat. The interface is designed to minimize the steps required to complete tasks, making it much faster than traditional personal assistants.

Elon MuskâĂŹs take: "WeâĂŹre headed toward a world where AI will assist us in almost everything." Musk's vision seems to come to life with assistants like Gemini, which promise a future where technology is seamlessly woven into our daily tasks.

5.3 AI for Creative Industries

5.3.1 Flux: AI for Photography

Editing photos has never been easier thanks to **Flux** [61], an AI-powered photography tool that can enhance your images with a single click. Whether you want to adjust lighting, remove unwanted objects, or apply artistic filters, Flux takes care of it all. The toolâĂŹs web interface is designed to be as simple as possible, offering sliders and one-click presets that instantly improve your photos.

How Flux is reshaping photography: With convolutional neural networks (CNNs) [8] as its backbone, Flux identifies elements like faces and objects, optimizing them for perfect lighting, balance, and color. This means even amateur photographers can achieve professional results in a fraction of

the time. The toolâĂŹs AI also suggests creative adjustments, turning ordinary shots into visually stunning images.

5.3.2 AI Coding Tools: The Future of Software Development

The world of programming is being revolutionized by AI tools designed to boost productivity and assist developers at every step. LetâĂŹs dive into three major players in the AI coding space: **Cursor** [62], **GitHub Copilot** [63], and **OpenDevin** [64].

Cursor: Cursor is your AI coding buddyâĂŤwhether youâĂŹre debugging or writing code, Cursor is there to help. It suggests code snippets, fixes bugs, and even generates entire functions based on your input, making coding more efficient than ever. The web GUI is streamlined, allowing developers to see real-time code suggestions and documentation, making it easy to implement solutions with minimal manual coding.

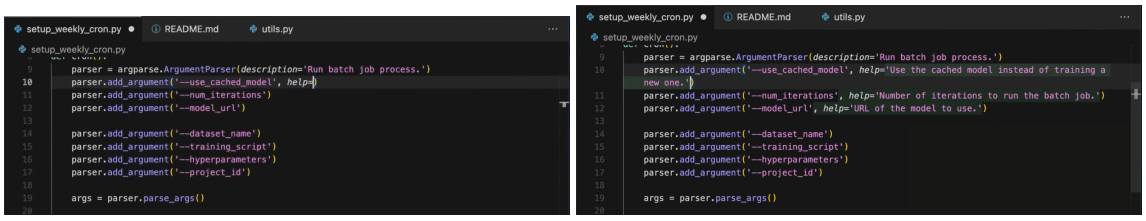


Figure 5.5: Left: Cursor analyzing code and suggesting fixes; Right: CursorâĂŹs detailed code generation based on minimal input. This AI tool can significantly reduce debugging and coding time.

GitHub Copilot: Powered by OpenAIâĂŹs Codex model, **GitHub Copilot** is like having a co-pilot while you code. It predicts what youâĂŹre trying to write, suggests code completions, and even writes boilerplate code, freeing you from repetitive tasks. The simplicity of GitHub CopilotâĂŹs interface means that even novice developers can easily integrate it into their workflow. By hovering over the suggestion box, you can apply entire blocks of code in one click, increasing efficiency dramatically.

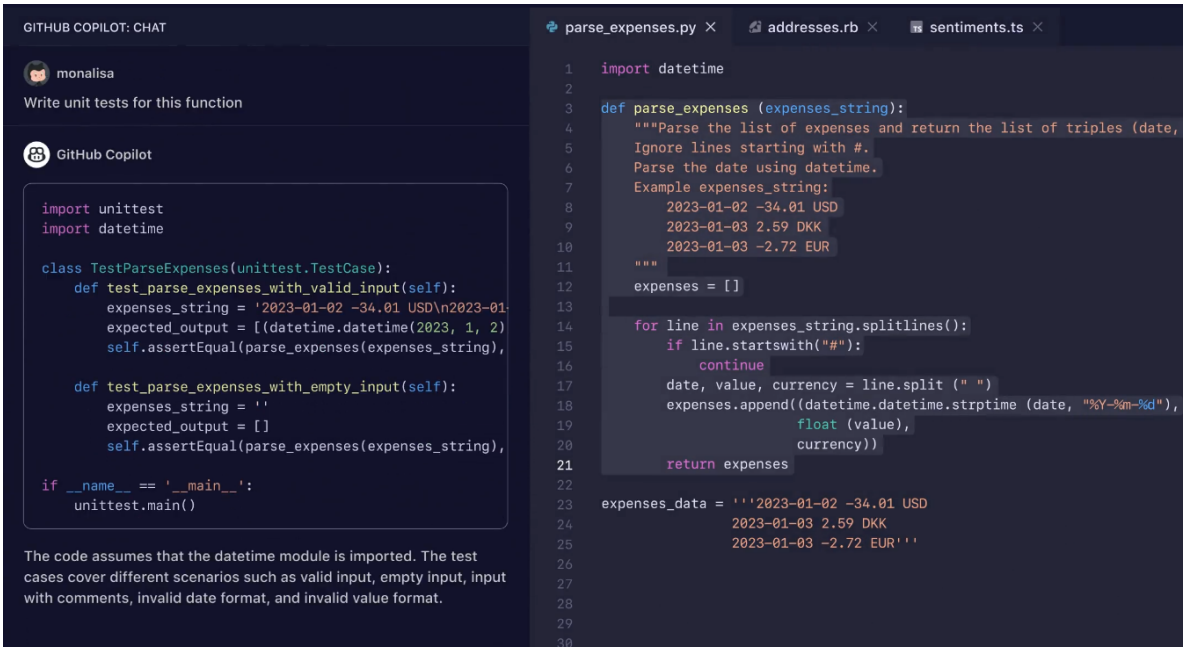


Figure 5.6: GitHub Copilot in action, suggesting code completions and writing functions automatically based on user prompts. This significantly reduces the time spent on writing repetitive code.

OpenDevin: Developed to provide smart insights, **OpenDevin** helps developers identify coding patterns, predict errors, and generate code documentation. Think of it as an intelligent assistant for ensuring that your code is not just functional but also optimized. OpenDevinâĂŹs intuitive dashboard offers a holistic view of your codebase, ensuring that every part of your project is analyzed for efficiency.

DevelopersâĂŹ Perspectives:

- Greg Brockman, co-founder of OpenAI, once said, "The future of programming is no programming." Tools like GitHub Copilot and Cursor are turning that vision into reality by doing the heavy lifting for developers.
- Andrej Karpathy, AI researcher and former director of AI at Tesla, has been vocal about the role AI will play in writing and maintaining code. "AI is fundamentally a new way of creating software."

5.4 AI in Geospatial Intelligence

5.4.1 Geospy.ai: Geospatial AI

Geospatial intelligence has become a critical part of environmental management, disaster response, and urban planning. **Geospy.ai** [65] uses AI to analyze satellite imagery, providing real-time insights into land use, forest cover, and even urban growth.

What Geospy.ai brings to the table: Geospy.ai is built on convolutional neural networks (CNNs) designed to process vast amounts of geospatial data. It can detect changes in infrastructure, track deforestation, and provide critical insights that would otherwise take years to gather manually.

Mark ZuckerbergâĂŹs view: Mark Zuckerberg once said, "In the future, every company will use AI." Geospy.ai embodies that idea by integrating AI into industries that might not seem obvious, like environmental conservation and city planning.

5.5 Summary

From ChatGPTâĂŹs conversations to Geospy.aiâĂŹs real-time satellite analysis, AI is advancing faster than ever, transforming industries and simplifying complex tasks. Developers, designers, and everyday users are benefiting from AI's ability to automate, assist, and inspire. Whether you're using AI to write code with tools like Cursor and GitHub Copilot or enhancing images with Flux, the future is bright, and AI is leading the charge.

Chapter 6

An Introduction to Generative art: A Beginner's Exploration

6.1 What are Diffusion models?

It is a type of probabilistic model that has been getting attention for its ability to generate high-quality, realistic images in the world of artificial intelligence (AI) and machine learning. In diffusion [66], a simple random noise distribution is gradually transformed into a meaningful structure or image over time, as a result of learning and refining. The diffusion process can be understood as the reverse of how physical particles spread or "diffuse" from an ordered state to a disordered one. As a core principle of AI, diffusion models are based on the reverse process of taking random noise and transforming it into detailed, structured data, such as an image.

In simple terms, diffusion models attempt to generate something meaningful (like an image) from randomness by learning how to reverse the natural process of noise. Through iterative refinement, they produce high-quality results that rival other AI methods, such as Generative Adversarial Networks (GANs) [27], but with greater stability and fewer generation problems.

6.2 Stable Diffusion: A Game-Changer in AI Art Generation

The Stable Diffusion [67] model is a state-of-the-art deep learning model designed for AI-based image generation. After its release in 2022, it was quickly recognized as a breakthrough in AI art. Previous models such as GANs and autoregressive models had shown promise in generating realistic images but often came with limitations such as instability, mode collapse (repeating the same type of output repeatedly), and limited generation capabilities.

Stable Diffusion, as its name suggests, brings greater stability and flexibility. It takes the diffusion process and applies it to generate high-resolution, coherent images from random noise, learning from massive datasets of images and their corresponding descriptions. What sets Stable Diffusion apart from other image-generation techniques is its ability to create highly detailed images, even when starting from simple prompts. It can generate artwork, landscapes, portraits, and even abstract imagery based on text descriptions, making it one of the most powerful tools in the AI art world.

The model works by iteratively refining noisy images, using a process called denoising diffusion probabilistic models (DDPM). Through this, it learns to reverse the noise and generate a clear, realistic

image from random patterns, effectively "stabilizing" the generation process.

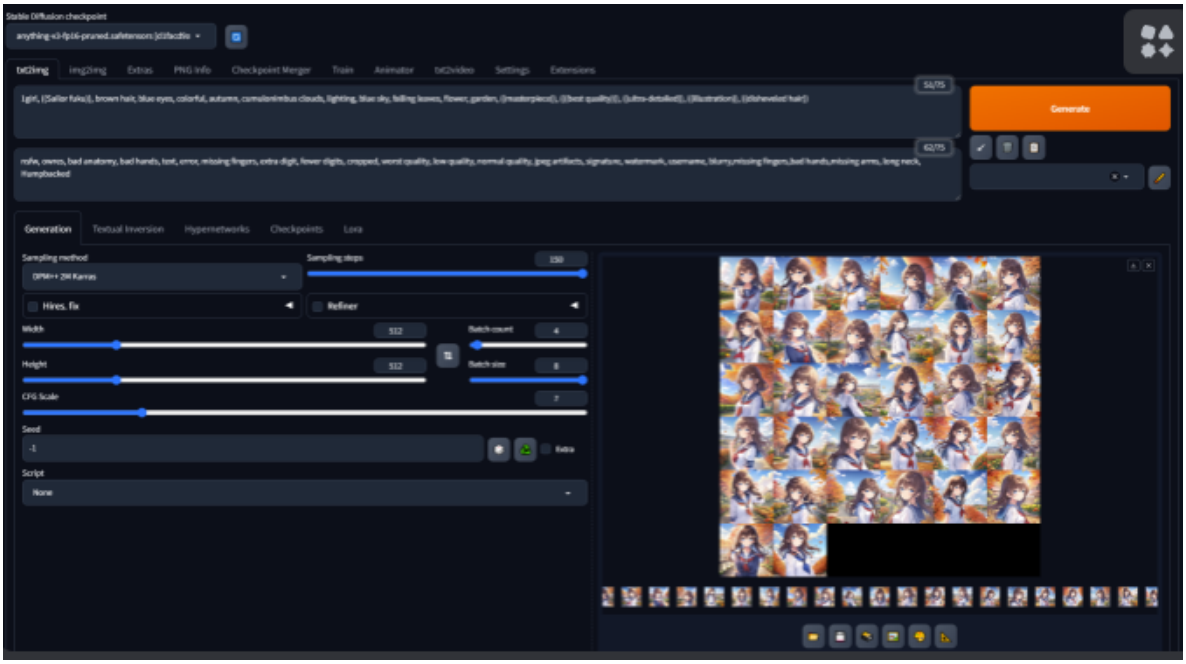


Figure 6.1: Snapshot of the image generation process using Stable Diffusion with WebUI. The settings and parameters used for generating high-quality images can be easily adjusted via the graphical interface.



Figure 6.2: Animated sequence showing how Stable Diffusion refines the noise iteratively to create coherent and high-quality images. These frames capture various stages of the image-generation process.

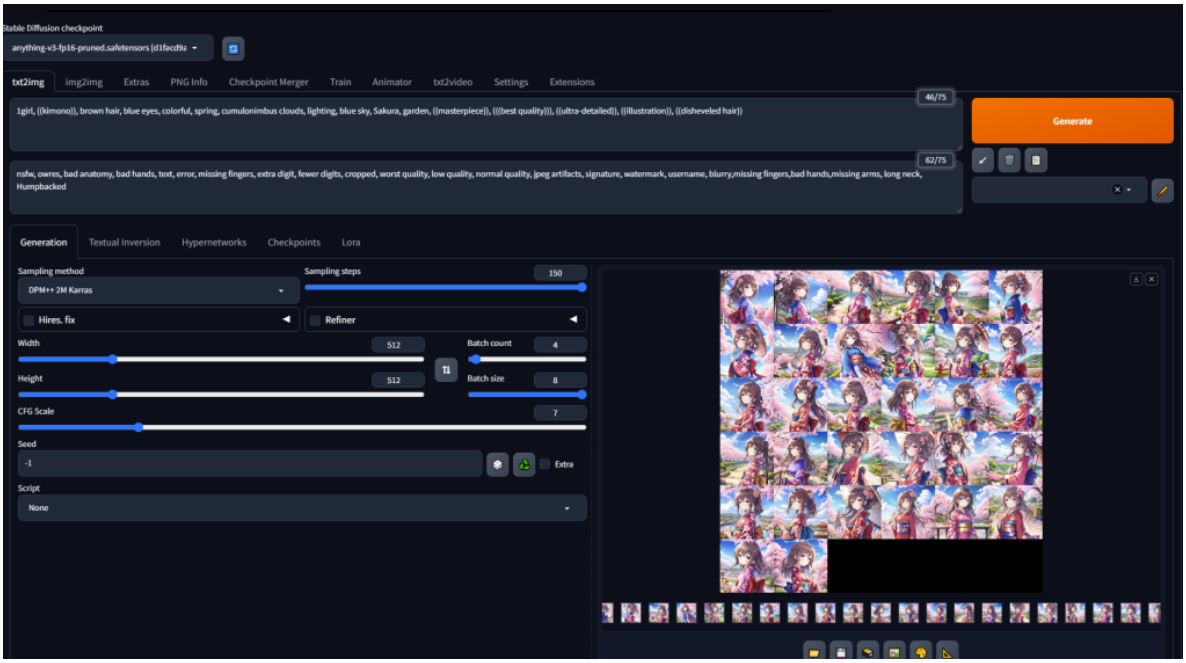


Figure 6.3: A demo of Stable Diffusion using the WebUI interface, showcasing its flexibility in creating multiple images at once with different prompts and settings.



Figure 6.4: Another set of animated frames from the Stable Diffusion process, showing how different details emerge during the denoising process to create intricate, final images.

6.3 AI Art: The Fusion of Creativity and Technology

AI art has revolutionized the way we think about creativity. Artists, designers, and hobbyists are now leveraging AI tools like Stable Diffusion to generate complex, imaginative artwork that previously required years of skill to produce. By feeding the model a textual description, users can see their ideas transformed into vivid imagery within minutes. This blend of human creativity and machine learning is shaping a new era of digital art, where AI acts as a collaborator, expanding the boundaries of artistic possibilities.

Artificial intelligence can create surreal, abstract compositions as well as hyper-realistic portraits and landscapes. The most exciting thing about this technology is that it allows anyone to realize their visions, regardless of their artistic background. As AI models become more refined, they also begin to learn individual artistic styles, giving creators greater control.

6.4 Two Main Interfaces: WebUI vs. ComfyUI

User interfaces (GUIs) are commonly used to make complex technology more accessible when using Stable Diffusion or other diffusion models to generate art. Two of the most popular interfaces are **WebUI** and **ComfyUI**. While both offer access to diffusion models for image generation, the needs and preferences of their users vary slightly.

6.4.1 WebUI: The User-Friendly Option

WebUI [68], short for "Web User Interface," is one of the most popular platforms for interacting with Stable Diffusion. ItâĂŹs known for being user-friendly and accessible to a broad audience, from beginners to more advanced users. WebUI is designed with simplicity in mind, providing an intuitive interface that allows users to easily input text prompts and generate images without needing extensive technical knowledge.

WebUIâĂŹs key features include:

- **Ease of Use**: A simple, clean design makes it easy to start generating images right away.
- **Customization**: Users can tweak various settings, such as image resolution, sampling steps, and denoising strength, to fine-tune their results.
- **Compatibility**: WebUI supports a variety of models, including the latest versions of Stable Diffusion, and integrates with plugins and extensions to offer more advanced functionality.

For many users, especially those just beginning to explore AI-generated art, WebUI is the ideal starting point. It provides the essential tools to generate stunning images quickly and with minimal technical barriers.

6.4.2 ComfyUI: Advanced Workflow Customization

While WebUI is excellent for ease of use, ComfyUI [69] is geared toward users who want more control over the generation process. ComfyUI allows for advanced workflow customization, where users can break down the image generation process into a series of interconnected steps or "nodes," similar to how you would work in a visual programming environment.

Some key features of ComfyUI include:

- **Node-based Workflow**: Users can create custom pipelines for image generation by connecting different processing steps. This allows for greater experimentation and fine-tuning.
- **Flexibility**: ComfyUI is designed for users who want to experiment with different diffusion models and algorithms. It offers more granular control over the image creation process, enabling users to manipulate every aspect of the generation pipeline.
- **Powerful Customization**: This platform is best suited for users who are comfortable with more technical aspects of image generation, allowing them to test various parameters and methods to achieve unique and precise results.

While ComfyUI has a steeper learning curve compared to WebUI, itâĂŹs a powerful tool for those who want to dive deeper into the mechanics of diffusion models and experiment with different workflows.

6.5 Comparison of WebUI and ComfyUI

The following table provides a comparison of the pros and cons of both WebUI and ComfyUI:

WebUI	ComfyUI
Pros	
• User-friendly interface, ideal for beginners • Simple setup and minimal technical knowledge required • Quick image generation with default settings • Supports various models and plugins • Customizable image parameters like resolution, sampling steps, and denoising	• Node-based workflow allows for precise control • Highly customizable pipelines for advanced users on detailed control over each generation step • Ideal for creating complex and specific workflows • Workflows are transferable and can be finalized as a Python module to run in an automatic pipeline.
Cons	
• Limited customization compared to more advanced tools • May not offer enough control for experienced users • Less suited for highly technical or experimental workflows • The number of users is shrinking	• Steeper learning curve, requires more technical understanding • Can be overwhelming for beginners • Longer setup and image generation process due to the complexity of the pipeline • May require more time to fine-tune and test parameters
*** It is strongly recommended that beginners focus their learning on ComfyUI**	

Table 6.1: Comparison of WebUI and ComfyUI: Pros and Cons

Since WebUI is relatively simple and user-friendly, it's a great starting point for those new to AI-generated art. By exploring WebUI first, users can quickly grasp the basic concepts of diffusion models and image generation without being overwhelmed by complex features. This foundational experience will help in understanding the more advanced functionalities that ComfyUI offers later on, making the transition smoother and more rewarding when moving to a more powerful and customizable tool like ComfyUI.

6.6 Getting Started with WebUI

WebUI is a graphic interface that lets users play with diffusion-based generative models easily and prototyping quickly. This section will walk through the process of installing WebUI and generating your first image.

6.6.1 Installation

To get started with WebUI, you will first need to download and install it. Follow these steps to install WebUI on your system:

1. **Download WebUI**: Head over to the official **WebUI GitHub repository** and clone the repository using the following command:

   ```
   git clone https://github.com/AUTOMATIC1111/stable-diffusion-webui
   ```

   After the clone, you can follow the instructions in**Installation and Running**. Keep in mind to match your operating system and graphics hardware.

2. **Launch WebUI**: Once installed, run the following command to launch WebUI on Linux:

   ```
   bash webui.sh
   ```

   For Windows users, you can launch it using:

   ```
   webui-user.bat
   Then the WebUI should be opening on your web browser.
   ```

6.6.2 Generate Your First Image

WebUI comes with a stable diffusion 1.5 checkpoint, to replace any version of the diffusion model, please follow this folder structure to put your diffusion generative model:

```
stable-diffusion-webui/
|-- models/ # Contains subfolders for different model types
| |-- Stable-diffusion/ # Stable Diffusion model checkpoint files
| |-- Lora/ # LoRA model files
| |-- VAE/ # Variational AutoEncoder models
| |...
```

Let's start with the basic usage: generate an image from text (txt2img tab).

- **Stable Diffusion Checkpoint**: In this dropdown, select the stable diffusion model you want to use from the `Stable-diffusion` folder.
- **Prompts**: Describe want you want to see in the `Prompt` text box. Within `Negative Prompt` text box, you can put things you don't want in the image.
- **Width and Height**: This is the size of the output image. Different diffusion models might have different output resolutions, set at least one side to 512 for v1.x stable-diffusion models, and 1024 for stable-diffusion xl models.
- **Sampling Methods**: Choose the algorithm for the denoising process. You can prototype from `DPM++ 2M Karras` [70] since it provides a good balance between speed and quality.
- **Sampling Steps**: The number of steps for the denoising algorithm above. When you choose different denoising algorithms the best number of steps might change. Tweak the sampling steps with the methods.

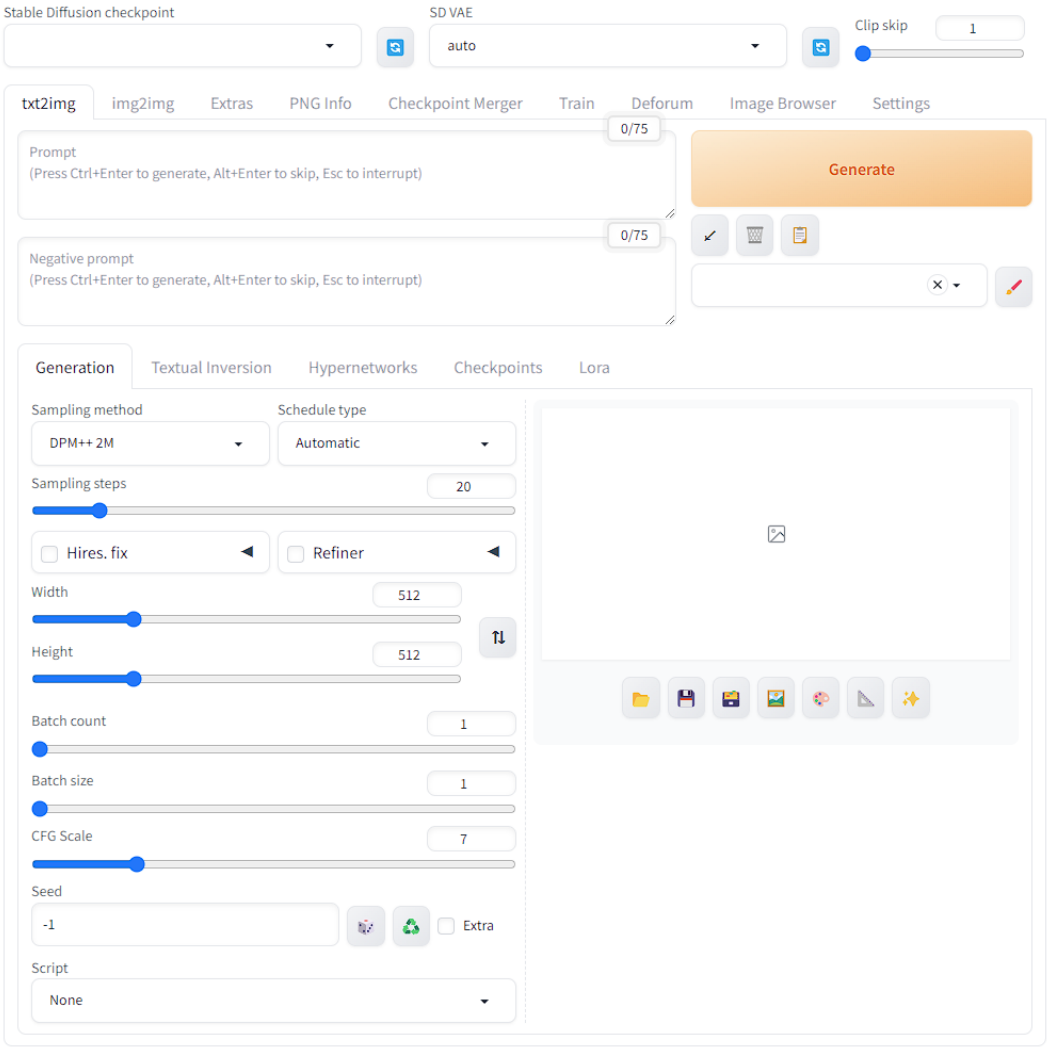


Figure 6.5: The WebUI Interface



Figure 6.6: Prompt: A highly detailed matte painting of huge sector v treehouse by Studio Ghibli, Makoto Shinkai, by Artgerm, by Greg Rutkowski, volumetric lighting, octane render, 4K resolution, trending on artstation, masterpiece

- **Batch Count**: Number of times you run the image generation process.
- **Batch Size**: Number of images to generate per pipeline run.

Once you choose your preferred parameters, click `Generate` then wait for your images to be generated. Remember, you can always leverage LLM to improve your prompts!

6.7 Getting Started with ComfyUI

ComfyUI is a powerful node-based interface designed for generating images using AI models like Stable Diffusion. This section will walk through the process of installing ComfyUI, setting up a basic workflow, and generating your first image.

6.7.1 Installation

To get started with ComfyUI, you will first need to download and install it. Follow these steps to install ComfyUI on your system:

1. **Download ComfyUI**: Head over to the official **ComfyUI GitHub repository** and clone the repository using the following command:

   ```
   git clone https://github.com/comfyanonymous/ComfyUI
   ```

   Alternatively, you can follow the **Read Me page** and download the standalone version for Windows and extract it.

2. **Install Dependencies**: If you are using Python, navigate to the ComfyUI directory and install the required dependencies using:

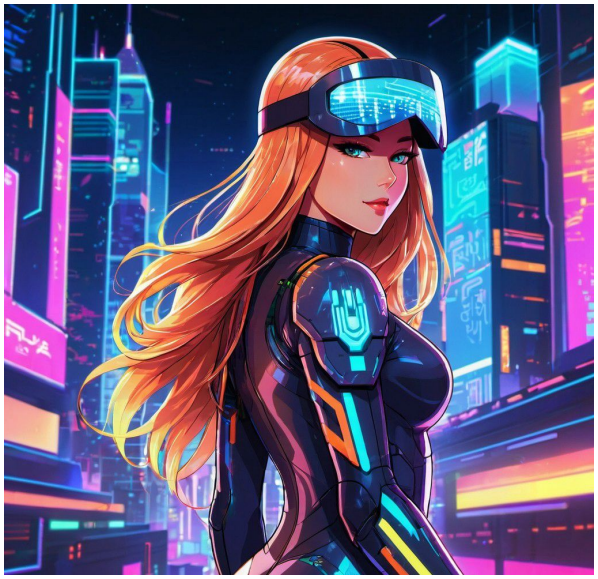


Figure 6.7: Prompt: A female character design, short hair wearing headphones, background in the city, Pixel style, transparent pvc jacket



Figure 6.8: Prompt: A female character design, short hair wearing headphones, background in the city, Harajuku Style, transparent pvc jacket

```
pip install -r requirements.txt
```

3. **Launch ComfyUI**: Once installed, run the following command to launch ComfyUI:

```
python main.py
```

For Mac users, you can launch it using:

```
python main.py --force-fp16
```

Try ComfyUI in Google Colab

Google Colab [71] (short for Collaboratory) is a free cloud-based service provided by Google which allows users to write and execute Python code in a Jupyter notebook environment. Colab particularly popular for data science, machine learning, and artificial intelligence projects because it provides access to powerful computational resources like GPUs and TPUs without needing a local setup.

To change your Google Colab notebook to use a GPU for free, follow these steps: In the Colab interface, click on `Runtime` in the menu bar. Then, select `Change runtime type` from the dropdown menu. In the dialog box that appears, under `Hardware accelerator`, choose `GPU` from the dropdown. After selecting it, click `Save`. This will enable the GPU for your session, allowing for faster computation, particularly for tasks involving machine learning or deep learning models.

Use this **Colab notebook** to clone the **ComfyUI** repo and install required packages. The same notebook can be found inside the **ComfyUI GitHub repository**. This is a well-annotated notebook, and what each cell is used for should be well-annotated and self-explanatory, keep in mind that your free GPU session may subject to early termination.

6.7.2 Install ComfyUI Manager

The **ComfyUI-Manager** is an extension designed to enhance the usability of ComfyUI. To install and use the ComfyUI Manager, follow these steps, which will guide you through the installation process and the basic usage of the manager. This will help you manage custom nodes in ComfyUI efficiently.

- First, make sure you have ComfyUI installed. If it's not installed yet, download it from the ComfyUI GitHub repository and follow the installation instructions.
- Open a terminal or command prompt, and navigate to the custom_nodes directory in your ComfyUI folder:

```
cd path/to/ComfyUI/custom_nodes/
```

- Clone the ComfyUI Manager repository into the custom_nodes directory using:

```
git clone https://github.com/ltdrdata/ComfyUI-Manager.git
```

- Restart ComfyUI to recognize the ComfyUI Manager. This can be done by closing and reopening the application.
- Once ComfyUI restarts, you can find the ComfyUI Manager on the interface. Click `Manager` in the main menu to open the management interface for custom nodes.

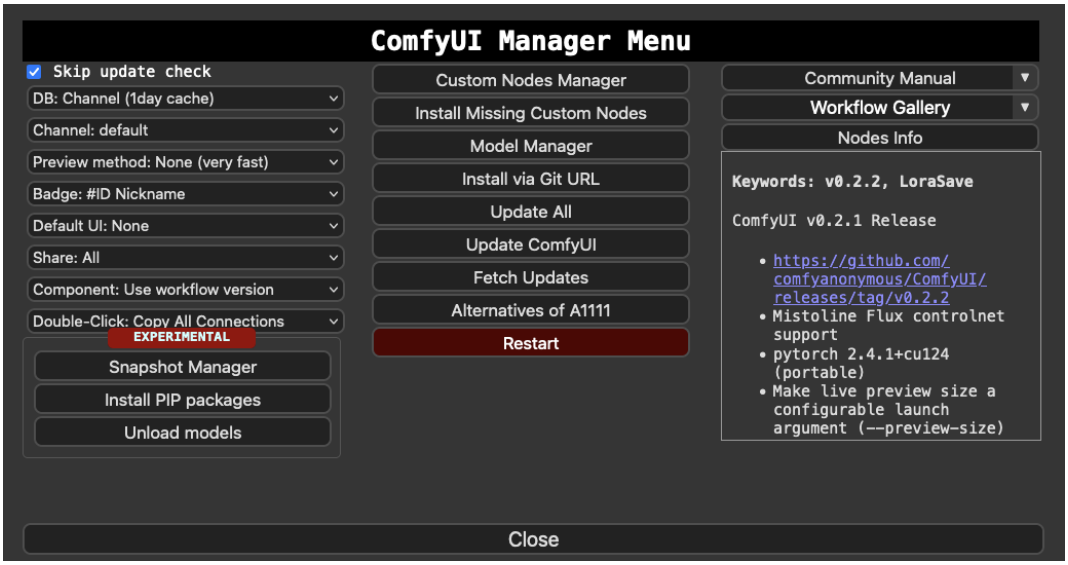


Figure 6.9: The ComfyUI Manager Menu

- In the ComfyUI Manager, you can use the `Install Custom Nodes` option to automatically detect and install any missing nodes. This feature is especially helpful when uploading new workflows, as it simplifies the process by listing missing nodes in red.
- Additionally, the Manager allows you to update, disable, or remove nodes. Click on `Update All` to refresh and ensure all nodes are up to date.
- If you come across conflicting nodes (shown in yellow), they can be disabled directly from the ComfyUI Manager interface to avoid issues.

6.7.3 Setting Up Your Default Workflow

The structure of the ComfyUI folder should look like the following:

```
ComfyUI/
|-- custom_nodes/ # Directory for custom node files
|-- input/ # Folder where uploaded assets are stored
|-- models/ # Contains subfolders for different model types
| |-- checkpoints/ # Stable Diffusion model checkpoint files
| |-- clip/ # CLIP model files
| |-- clip_vision/ # CLIP Vision model files
| |-- controlnet/ # ControlNet models
| |-- loras/ # LoRA model files
| |-- vae/ # Variational AutoEncoder models
| |-- unet/ # UNet architecture files
| `-- upscale_models/ # Models for upscaling images
|-- notebooks/ # Jupyter Notebooks for Colab and remote executions
|-- output/ # Generated images saved here
|-- config/ # Configuration files for models and environment
|-- extra_model_paths.yaml.example # Example config file for extra model paths
|-- comfyui.log # Runtime log file
|-- README_VERY_IMPORTANT.txt # Instructions and usage details
`-- run_nvidia_gpu.bat # Script to run ComfyUI on Nvidia GPUs
```

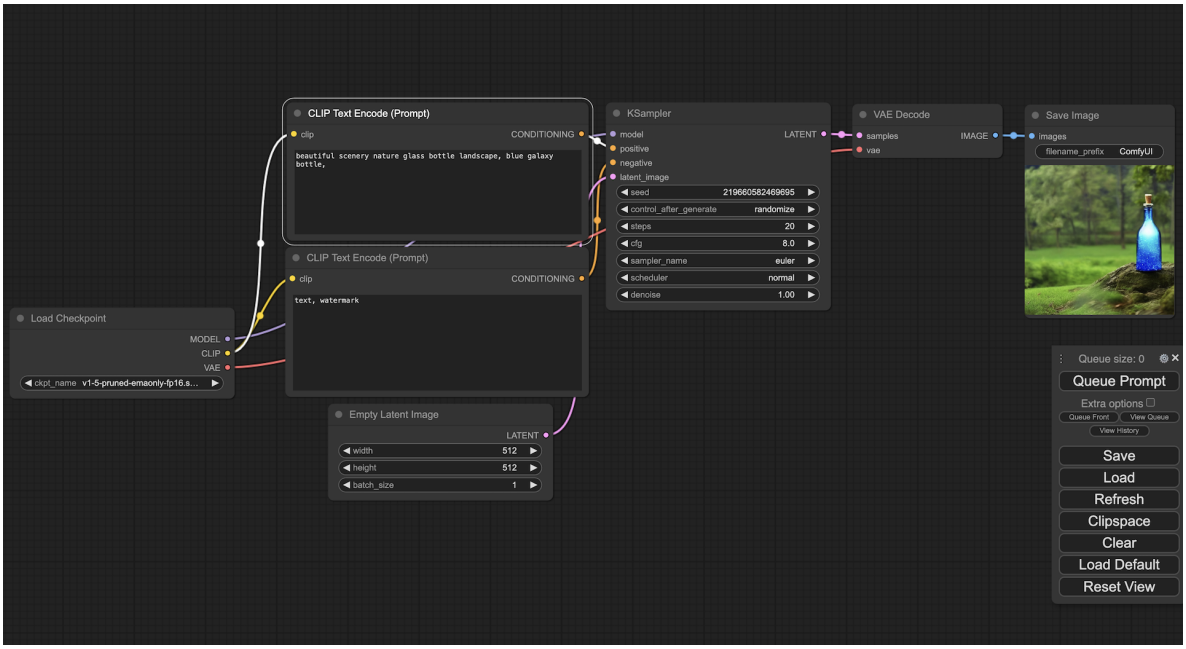


Figure 6.10: The ComfyUI interface with a simple image generation workflow

Once ComfyUI is installed, you can load a default workflow for generating images. Here's how:

1. **Download a Model**: Download a model checkpoint, such as Stable Diffusion v1.5, and place it in the directory `ComfyUI/models/checkpoints`. **Civitai** [72] is a community-driven platform for sharing and discovering custom models and assets used in AI-generated art, primarily focused on Stable Diffusion and other diffusion models.
2. **Load the Default Workflow**: Open ComfyUI and click the `Load Default` button to load the basic workflow. In the `Load Checkpoint` node, select the model you just downloaded.

6.7.4 Generate Your First Image

ComfyUI uses a node-based interface (see **Figure 6.10**) where all operations are performed by connecting different nodes. To begin generating images:

- **Add Nodes**: Right-click on the canvas and select `Add Nodes`, or double left-click the canvas and search for a node. For image generation, important nodes include `Load Checkpoint`, `CLIP Text Encode`, and `KSampler`.

- **Prompts and Parameters**: Set your prompts in the `CLIP Text Encode` node (positive and negative prompt nodes are connected to `KSampler` respectively) to guide image generation. Adjust parameters such as the number of diffusion `steps` and the `CFG scale` in the `KSampler` node.

- **Queue and Generate**: Once the nodes are connected, click `Queue Prompt` to generate the image. Experiment with different prompts to refine the results.

6.7.5 Try Other Models

To use other models in ComfyUI, follow these step-by-step instructions to ensure proper installation and workflow setup (this example will show you how to use the Anything XL model):

- **Download the Anything XL Model**: First, download the Anything XL model from a platform like Hugging Face or CivitAI. Make sure the file is in .safetensors or .ckpt format.
- **Place the Model in the Correct Directory**: Navigate to your ComfyUI installation directory, and place the downloaded Anything XL model in the ComfyUI/models/checkpoints folder.
- **Launch ComfyUI**: Open ComfyUI by running the appropriate batch file depending on your system (e.g., `run_nvidia_gpu.bat` for NVIDIA GPUs).
- **Clear the Workspace**: Once ComfyUI opens, click `Clear` to clear any pre-loaded workflows on the workspace. This will give you a clean slate to start from scratch.
- **Load the Anything XL Model**: Right-click on the workspace, and navigate to `Add Node > Loaders > Load Checkpoint`. From the options in the dropdown, select the Anything XL model you placed in the checkpoints folder earlier.
- **Add the KSampler Node**: Right-click on the workspace again and go to `Add Node > Sampling > KSampler`. This node is essential for generating images from random noise.
- **Connect the Model and KSampler**: Connect the output from the `Load Checkpoint` node (MODEL) to the input of the `KSampler` node (model). This links the model to the sampling process.
- **Set Up Prompts (Positive/Negative)**: To tell the model what to generate, you need to add prompt nodes. Right-click the workspace and select `Add Node > Conditioning > CLIP Text Encode (Prompt)`. This will be your positive prompt node. Repeat this step for the negative prompt. Link the positive prompt's output to the `conditioning` input of the KSampler node. Similarly, link the negative prompt's output to the `conditioning (negative)` input on the KSampler.
- **Add Empty Latent Image**: Right-click on the workspace and select `Add Node > latent > Empty Latent Image`. This node will act as the starting point for the image generation. Connect its output to the latent image input on the KSampler.
- **VAE Decode**: The generated latent image needs to be decoded into a viewable format. Right-click and select `Add Node > latent > VAE Decode`. Connect the KSampler's latent output to the `samples` input on the VAE Decode node.
- **Save the Image**: Add the final node by selecting `Add Node > image > Save Image`. This node will save the generated image to your output folder. Connect the output of the VAE Decode node to the Save Image node.
- **Adjust Parameters**: You can now fine-tune your settings. Adjust the image size in the Empty Latent Image node to match Anything XLâĂŹs training resolution (1024x1024). You can also modify the sampling steps and CFG scale in the KSampler node to refine the output.
- **Generate the Image**: Once all nodes are connected, and parameters are set, click `Queue Prompt` to generate the image. The process might take a few minutes, depending on your system and settings.
- **Review and Save**: After the image is generated, it will be saved automatically in the output folder, but you can preview and make any additional adjustments to your prompts or settings if needed.

When using the Anything XL model with ComfyUI, configuring the KSampler node correctly quite important. Here's an overview of the recommended KSampler settings, along with an explanation of their functions:

- **Steps**: The number of sampling steps. For Anything XL, you typically want to set this to around 20-40 steps. Higher steps usually result in better image quality but will take longer to process. For quick

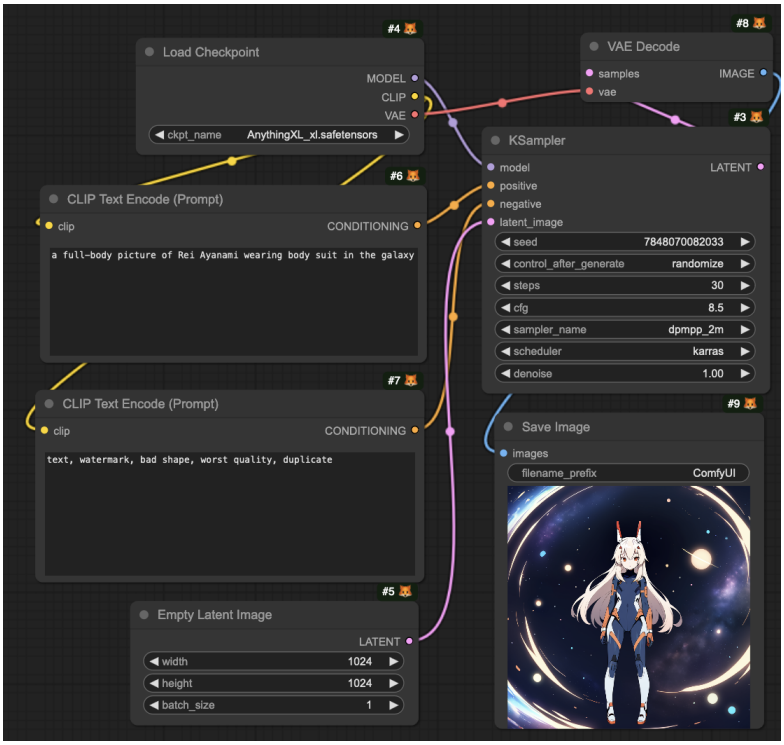


Figure 6.11: The ComfyUI interface with a simple image generation workflow

previews, you might start with a lower value like 20, but for high-quality final outputs, going up to 40 or even 50 steps can produce more refined images.

- **CFG** (Classifier-Free Guidance) Scale: This parameter controls how much the model adheres to the text prompts. For Anything XL, a good starting value is 7-12. Lower values like 7 will give more creative freedom to the model, while higher values like 12 will enforce stricter adherence to the prompt. Start with 7.5 for balanced results, and adjust based on your needs.
- **Seed**: This is used for deterministic image generation. By setting a specific seed, you can regenerate the exact same image. If you want random images each time, leave this as -1 (which stands for random seed).
- **Sampler Algorithm**: KSampler offers different algorithms to control how the noise is removed over time. For Anything XL, Euler A or DPM++ 2M Karras are popular choices for producing sharper and more detailed results. Euler A is faster but might not be as precise as DPM++ 2M Karras. If you prioritize speed, go with Euler A, otherwise try DPM++ 2M Karras for high-quality outputs.
- **Denoising Strength**: This parameter is especially useful if you're using techniques like Img2Img or refining an existing image. A value around 0.7-0.9 works well for generating detailed results. For text-to-image generation, you usually leave this at 1.

You can checkout the example in **Figure 6.11**

6.7.6 Advanced Features

ComfyUI also supports more advanced workflows, such as image-to-image (Img2Img) generation, where you can input an existing image to modify. Additionally, you can explore various node shortcuts like `CTRL + drag` to select multiple nodes, or `SHIFT + drag` to move them.

This setup will allow you to get started with ComfyUI and generate your first batch of AI-powered images.

6.8 Summary

As we explored in this chapter, diffusion models and their roles in AI-driven image generation were the foundations of generative art. Diffusion models have emerged as a powerful method for transforming random noise into coherent, high-quality images, providing a stable alternative to earlier techniques like Generative Adversarial Networks (GANs). The development of models like Stable Diffusion has revolutionized AI art generation, enabling artists and hobbyists alike to create detailed and vivid imagery from simple text prompts.

We also examined two main interfacesâĂŤWebUI and ComfyUIâĂŤthrough which users can interact with diffusion models. WebUI, with its user-friendly design, offers an accessible platform for beginners to generate AI art with minimal technical knowledge. On the other hand, ComfyUI is tailored for more advanced users, providing granular control over the generation process through a node-based workflow, allowing for greater experimentation and customization.

The chapter showed how AI art merges creativity and technology in a dynamic way. As AI continues to push the boundaries of what's possible in the world of art, even people without traditional artistic backgrounds can explore new realms of digital creativity with tools like Stable Diffusion.

Appendix A

Image Demos

This appendix includes various image demos that illustrate different aspects of the content discussed in the book.



Figure A.1: Bird Demo. These images showcase generative models creating realistic representations of animals.



Figure A.2: Cat Demo. These images showcase generative models creating realistic representations of animals.



Figure A.3: House Demo. Examples of generative models producing architectural and still life images via prompt: Beautiful lake view villa



Figure A.4: Fruit Demo. Examples of generative models producing architectural and still life images via prompt: Various fruits on the table and Fruit platter



Figure A.5: Animation Demo 1. These frames illustrate how the model generates animations.Promt:1girl, ((Sailor fuku)), brown hair, blue eyes, colorful, autumn, cumulonimbus clouds, lighting, blue sky, falling leaves, flower, garden, ((masterpiece)), (((best quality))), ((ultra-detailed)), ((illustration)), ((disheveled hair))



Figure A.6: Animation Demo 2. These frames illustrate how the model generates animations. prompt: 1girl, brown hair, green eyes, colorful, autumn, cumulonimbus clouds, lighting, blue sky, falling leaves, garden



Figure A.7: Animation Demo 3. Further frames from the animation generation process. Prompt: 1girl, brown hair, blue eyes, colorful, autumn, cumulonimbus clouds, lighting, blue sky, falling leaves, garden



Figure A.8: Animation Demo 4. Further frames from the animation generation process.Prompt:1girl, brown hair, blue eyes, colorful, autumn, cumulonimbus clouds, lighting, blue sky, falling leaves, garden



Figure A.9: Animation Demo 4. Further frames from the animation generation process.Prompt:1girl, ((Sailor fuku)), brown hair, blue eyes, colorful, autumn, cumulonimbus clouds, lighting, blue sky, falling leaves, flower, garden, ((masterpiece)), (((best quality))), ((ultra-detailed)), ((illustration)), ((disheveled hair))



Figure A.10: Animation Demo 6. The final frames in the animation sequence.Prompt: 1girl, ((kimono)), brown hair, blue eyes, colorful, spring, cumulonimbus clouds, lighting, blue sky, Sakura, garden, ((masterpiece)), (((best quality))), ((ultra-detailed)), ((illustration)), ((disheveled hair))

Bibliography

[1] R. Gruetzemacher and J. Whittlestone, "The transformative potential of artificial intelligence," *Futures*, vol. 135, p. 102884, 2022.

[2] L. Veldkamp, "Valuing Data as an Asset*," *Review of Finance*, vol. 27, pp. 1545–1562, 01 2023.

[3] O. Alghamdi and G. Agag, "Competitive advantage: A longitudinal analysis of the roles of data-driven innovation capabilities, marketing agility, and market turbulence," *Journal of Retailing and Consumer Services*, vol. 76, p. 103547, 2024.

[4] H. A. Selmy, H. K. Mohamed, and W. Medhat, "Big data analytics deep learning techniques and applications: A survey," *Information Systems*, vol. 120, p. 102318, 2024.

[5] I. H. Sarker, "Deep learning: A comprehensive overview on techniques, taxonomy, applications and research directions," *SN Computer Science*, vol. 2, no. 6, p. 420, 2021. Âl' The Author(s), under exclusive licence to Springer Nature Singapore Pte Ltd 2021.

[6] C. Chakraborty, M. Bhattacharya, S. Pal, and S.-S. Lee, "From machine learning to deep learning: Advances of the recent data-driven paradigm shift in medicine and healthcare," *Current Research in Biotechnology*, vol. 7, p. 100164, 2024.

[7] M. M. Taye, "Understanding of machine learning with deep learning: Architectures, workflow, applications and future directions," *Computers*, vol. 12, no. 5, 2023.

[8] K. O'Shea and R. Nash, "An introduction to convolutional neural networks," 2015.

[9] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," 2023.

[10] R. M. Schmidt, "Recurrent neural networks (rnns): A gentle introduction and overview," 2019.

[11] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, pp. 1735–80, 12 1997.

[12] A. Radford, K. Narasimhan, T. Salimans, and I. Sutskever, "Improving language understanding by generative pre-training," *OpenAI*, 2018.

[13] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," 2019.

[14] T. Wolf, L. Debut, V. Sanh, J. Chaumond, C. Delangue, A. Moi, P. Cistac, T. Rault, R. Louf, M. Funtowicz, J. Davison, S. Shleifer, P. von Platen, C. Ma, Y. Jernite, J. Plu, C. Xu, T. Le Scao, S. Gugger, M. Drame, Q. Lhoest, and A. Rush, "Transformers: State-of-the-art natural language processing,"

in *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations* (Q. Liu and D. Schlangen, eds.), (Online), pp. 38–45, Association for Computational Linguistics, Oct. 2020.

[15] OpenAI, J. Achiam, S. Adler, S. Agarwal, L. Ahmad, I. Akkaya, F. L. Aleman, D. Almeida, J. Altenschmidt, S. Altman, S. Anadkat, R. Avila, I. Babuschkin, S. Balaji, V. Balcom, P. Baltescu, H. Bao, M. Bavarian, J. Belgum, I. Bello, J. Berdine, G. Bernadett-Shapiro, C. Berner, L. Bogdonoff, O. Boiko, M. Boyd, A.-L. Brakman, G. Brockman, T. Brooks, M. Brundage, K. Button, T. Cai, R. Campbell, A. Cann, B. Carey, C. Carlson, R. Carmichael, B. Chan, C. Chang, F. Chantzis, D. Chen, S. Chen, R. Chen, J. Chen, M. Chen, B. Chess, C. Cho, C. Chu, H. W. Chung, D. Cummings, J. Currier, Y. Dai, C. Decareaux, T. Degry, N. Deutsch, D. Deville, A. Dhar, D. Dohan, S. Dowling, S. Dunning, A. Ecoffet, A. Eleti, T. Eloundou, D. Farhi, L. Fedus, N. Felix, S. P. Fishman, J. Forte, I. Fulford, L. Gao, E. Georges, C. Gibson, V. Goel, T. Gogineni, G. Goh, R. Gontijo-Lopes, J. Gordon, M. Grafstein, S. Gray, R. Greene, J. Gross, S. S. Gu, Y. Guo, C. Hallacy, J. Han, J. Harris, Y. He, M. Heaton, J. Heidecke, C. Hesse, A. Hickey, W. Hickey, P. Hoeschele, B. Houghton, K. Hsu, S. Hu, X. Hu, J. Huizinga, S. Jain, S. Jain, J. Jang, A. Jiang, R. Jiang, H. Jin, D. Jin, S. Jomoto, B. Jonn, H. Jun, T. Kaftan, ÅĄukasz Kaiser, A. Kamali, I. Kanitscheider, N. S. Keskar, T. Khan, L. Kilpatrick, J. W. Kim, C. Kim, Y. Kim, J. H. Kirchner, J. Kiros, M. Knight, D. Kokotajlo, ÅĄukasz Kondraciuk, A. Kondrich, A. Konstantinidis, K. Kosic, G. Krueger, V. Kuo, M. Lampe, I. Lan, T. Lee, J. Leike, J. Leung, D. Levy, C. M. Li, R. Lim, M. Lin, S. Lin, M. Litwin, T. Lopez, R. Lowe, P. Lue, A. Makanju, K. Malfacini, S. Manning, T. Markov, Y. Markovski, B. Martin, K. Mayer, A. Mayne, B. McGrew, S. M. McKinney, C. McLeavey, P. McMillan, J. McNeil, D. Medina, A. Mehta, J. Menick, L. Metz, A. Mishchenko, P. Mishkin, V. Monaco, E. Morikawa, D. Mossing, T. Mu, M. Murati, O. Murk, D. MÃľly, A. Nair, R. Nakano, R. Nayak, A. Neelakantan, R. Ngo, H. Noh, L. Ouyang, C. O'Keefe, J. Pachocki, A. Paino, J. Palermo, A. Pantuliano, G. Parascandolo, J. Parish, E. Parparita, A. Passos, M. Pavlov, A. Peng, A. Perelman, F. de Avila Belbute Peres, M. Petrov, H. P. de Oliveira Pinto, Michael, Pokorny, M. Pokrass, V. H. Pong, T. Powell, A. Power, B. Power, E. Proehl, R. Puri, A. Radford, J. Rae, A. Ramesh, C. Raymond, F. Real, K. Rimbach, C. Ross, B. Rotsted, H. Roussez, N. Ryder, M. Saltarelli, T. Sanders, S. Santurkar, G. Sastry, H. Schmidt, D. Schnurr, J. Schulman, D. Selsam, K. Sheppard, T. Sherbakov, J. Shieh, S. Shoker, P. Shyam, S. Sidor, E. Sigler, M. Simens, J. Sitkin, K. Slama, I. Sohl, B. Sokolowsky, Y. Song, N. Staudacher, F. P. Such, N. Summers, I. Sutskever, J. Tang, N. Tezak, M. B. Thompson, P. Tillet, A. Tootoonchian, E. Tseng, P. Tuggle, N. Turley, J. Tworek, J. F. C. Uribe, A. Vallone, A. Vijayvergiya, C. Voss, C. Wainwright, J. J. Wang, A. Wang, B. Wang, J. Ward, J. Wei, C. Weinmann, A. Welihinda, P. Welinder, J. Weng, L. Weng, M. Wiethoff, D. Willner, C. Winter, S. Wolrich, H. Wong, L. Workman, S. Wu, J. Wu, M. Wu, K. Xiao, T. Xu, S. Yoo, K. Yu, Q. Yuan, W. Zaremba, R. Zellers, C. Zhang, M. Zhang, S. Zhao, T. Zheng, J. Zhuang, W. Zhuk, and B. Zoph, "Gpt-4 technical report," 2024.

[16] S. Shahriar and K. Hayawi, "LetâĂŹs have a chat! a conversation with chatgpt: Technology, applications, and limitations," *Artificial Intelligence and Applications*, vol. 2, p. 11âĂŞ20, June 2023.

[17] T. Chen and C. Guestrin, "Xgboost: A scalable tree boosting system," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, vol. 11 of *KDD âĂŹ16*, p. 785âĂŞ794, ACM, Aug. 2016.

[18] R. Bain, C. Lynch, D. McDonnell, and K. Witheephanich, "An xgboost approach for industrial component degradation classification," in *2022 33rd Irish Signals and Systems Conference (ISSC)*, pp. 1–7, 2022.

[19] M. Niazkar, A. Menapace, B. Brentan, R. Piraei, D. Jimenez, P. Dhawan, and M. Righetti, "Applications of xgboost in water resources engineering: A systematic literature review (dec 2018âĂŞmay 2023)," *Environmental Modelling & Software*, vol. 174, p. 105971, 2024.

[20] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, Y. Chen, T. Lillicrap, F. Hui, L. Sifre, G. van den Driessche, T. Graepel, and D. Hassabis, "Mastering the game of go without human knowledge," *Nature*, vol. 550, no. 7676, pp. 354–359, 2017.

[21] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," 2015.

[22] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," in *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)* (J. Burstein, C. Doran, and T. Solorio, eds.), (Minneapolis, Minnesota), pp. 4171–4186, Association for Computational Linguistics, June 2019.

[23] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," 2016.

[24] J. Terven, D.-M. CÃşrdova-Esparza, and J.-A. Romero-GonzÃąlez, "A comprehensive review of yolo architectures in computer vision: From yolov1 to yolov8 and yolo-nas," *Machine Learning and Knowledge Extraction*, vol. 5, p. 1680âĂŞ1716, Nov. 2023.

[25] J. Jumper, R. Evans, A. Pritzel, T. Green, M. Figurnov, O. Ronneberger, K. Tunyasuvunakool, R. Bates, A. Å¡Ãŋdek, A. Potapenko, A. Bridgland, C. Meyer, S. A. A. Kohl, A. J. Ballard, A. Cowie, B. Romera-Paredes, S. Nikolov, R. Jain, J. Adler, T. Back, S. Petersen, D. Reiman, E. Clancy, M. Zielinski, M. Steinegger, M. Pacholska, T. Berghammer, S. Bodenstein, D. Silver, O. Vinyals, A. W. Senior, K. Kavukcuoglu, P. Kohli, and D. Hassabis, "Highly accurate protein structure prediction with alphafold," *Nature*, vol. 596, pp. 583–589, aug 2021.

[26] AlphaFold Protein Structure Database, "Image of protein structure." https://alphafold.ebi.ac.uk/, 2024. Accessed: 2024-09-22.

[27] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial networks," 2014.

[28] A. Creswell, T. White, V. Dumoulin, K. Arulkumaran, B. Sengupta, and A. A. Bharath, "Generative adversarial networks: An overview," *IEEE Signal Processing Magazine*, vol. 35, p. 53âĂŞ65, Jan. 2018.

[29] Google Developers, "Gan structure [image]." https://developers.google.com/machine-learning/gan/gan_structure, 2024. Accessed: 2024-09-22.

[30] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," 2013.

[31] M. Hausknecht and P. Stone, "Deep recurrent q-learning for partially observable mdps," 2017.

[32] Z. Wang, T. Schaul, M. Hessel, H. van Hasselt, M. Lanctot, and N. de Freitas, "Dueling network architectures for deep reinforcement learning," 2016.

[33] H. van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double q-learning," 2015.

[34] K. Chen, Z. Bi, Q. Niu, J. Liu, B. Peng, S. Zhang, M. Liu, M. Li, X. Pan, J. Xu, J. Wang, and P. Feng, "Deep learning and machine learning, advancing big data analytics and management: Tensorflow pretrained models," 2024.

[35] J. Ferreira, F. Almeida, and J. Monteiro, "Building an effective data warehousing for financial sector," *Automatic Control and Information Sciences*, vol. 3, p. 16âĂŞ25, Aug. 2017.

[36] Y. N. Silva, I. Almeida, and M. Queiroz, "Sql: From traditional databases to big data," in *Proceedings of the 47th ACM Technical Symposium on Computing Science Education*, SIGCSE '16, (New York, NY, USA), p. 413âĂŞ418, Association for Computing Machinery, 2016.

[37] A. Gupta, S. Tyagi, N. Panwar, S. Sachdeva, and U. Saxena, "Nosql databases: Critical analysis and comparison," in *2017 International Conference on Computing and Communication Technologies for Smart Nation (IC3TSN)*, pp. 293–299, 2017.

[38] Y. Tian, "The world of graph databases from an industry perspective," 2022.

[39] C. Gao, Y. Zheng, N. Li, Y. Li, Y. Qin, J. Piao, Y. Quan, J. Chang, D. Jin, X. He, and Y. Li, "A survey of graph neural networks for recommender systems: Challenges, methods, and directions," *ACM Trans. Recomm. Syst.*, vol. 1, mar 2023.

[40] K. Shvachko, H. Kuang, S. Radia, and R. Chansler, "The hadoop distributed file system," in *2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST)*, pp. 1–10, 2010.

[41] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica, "Spark: cluster computing with working sets," in *Proceedings of the 2nd USENIX Conference on Hot Topics in Cloud Computing*, HotCloud'10, (USA), p. 10, USENIX Association, 2010.

[42] D. Mazzei and R. Ramjattan, "Machine learning for industry 4.0: A systematic review using deep learning-based topic modelling," *Sensors*, vol. 22, no. 22, 2022.

[43] K. Choudhary, B. DeCost, C. Chen, A. Jain, F. Tavazza, R. Cohn, C. W. Park, A. Choudhary, A. Agrawal, S. J. L. Billinge, E. Holm, S. P. Ong, and C. Wolverton, "Recent advances and applications of deep learning methods in materials science," *npj Computational Materials*, vol. 8, no. 1, p. 59, 2022.

[44] OpenAI, "Chatgpt: September 2023 version," 2023. Large language model.

[45] P. S. Foundation, "Python language reference, version 3.x." https://www.python.org/, 2023. Available at https://www.python.org/.

[46] T. P. P. Authority, "pip," 2023. Version 23.1.2.

[47] M. Waskom, "Seaborn: Statistical data visualization," 2021. Version 0.11.2.

[48] J. D. Hunter, "Matplotlib: A 2d graphics environment," *Computing in Science & Engineering*, vol. 9, no. 3, pp. 90–95, 2007.

[49] M. Waskom, "Python tips dataset." https://github.com/mwaskom/seaborn-data, 2023. Available in the Seaborn-data repository.

[50] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and É. Duchesnay, "Scikit-learn: Machine learning in python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[51] B. Efron, T. Hastie, I. Johnstone, and R. Tibshirani, "Least angle regression," *The Annals of Statistics*, vol. 32, no. 2, pp. 407 – 499, 2004.

[52] A. Schneider, G. Hommel, and M. Blettner, "Linear regression analysis: Part 14 of a series on evaluation of scientific publications," *Deutsches ÃĎrzteblatt International*, vol. 107, pp. 776–782, November 2010.

[53] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015. Software available from tensorflow.org.

[54] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "Pytorch: An imperative style, high-performance deep learning library," *Advances in Neural Information Processing Systems*, vol. 32, pp. 8024–8035, 2019.

[55] L. Deng, "The mnist database of handwritten digit images for machine learning research [best of the web]," *IEEE Signal Processing Magazine*, vol. 29, no. 6, pp. 141–142, 2012.

[56] Ultralytics, "Yolov8: State-of-the-art object detection models." https://github.com/ultralytics/ultralytics, 2023. Version 8.0.

[57] A. Kirillov, E. Mintun, N. Ravi, H. Mao, C. Rolland, L. Gustafson, T. Xiao, S. Whitehead, A. C. Berg, W.-Y. Lo, P. DollÃąr, and R. Girshick, "Segment anything," 2023.

[58] A. PÃľrez-Vereda, R. Hervas, and C. Canal, "Digital avatars: A programming framework for personalized human interactions through virtual profiles," *Pervasive and Mobile Computing*, vol. 87, p. 101718, 11 2022.

[59] G. Yenduri, R. M, C. S. G, S. Y, G. Srivastava, P. K. R. Maddikunta, D. R. G, R. H. Jhaveri, P. B, W. Wang, A. V. Vasilakos, and T. R. Gadekallu, "Generative pre-trained transformer: A comprehensive review on enabling technologies, potential applications, emerging challenges, and future directions," 2023.

[60] G. Team, R. Anil, S. Borgeaud, J.-B. Alayrac, J. Yu, R. Soricut, J. Schalkwyk, A. M. Dai, A. Hauth, K. Millican, D. Silver, M. Johnson, I. Antonoglou, J. Schrittwieser, A. Glaese, J. Chen, E. Pitler, T. Lillicrap, A. Lazaridou, O. Firat, J. Molloy, M. Isard, P. R. Barham, T. Hennigan, B. Lee, F. Viola, M. Reynolds, Y. Xu, R. Doherty, E. Collins, C. Meyer, E. Rutherford, E. Moreira, K. Ayoub, M. Goel, J. Krawczyk, C. Du, E. Chi, H.-T. Cheng, E. Ni, P. Shah, P. Kane, B. Chan, M. Faruqui, A. Severyn, H. Lin, Y. Li, Y. Cheng, A. Ittycheriah, M. Mahdieh, M. Chen, P. Sun, D. Tran, S. Bagri, B. Lakshminarayanan, J. Liu, A. Orban, F. GÃijra, H. Zhou, X. Song, A. Boffy, H. Ganapathy, S. Zheng,

H. Choe, ÃĄgoston Weisz, T. Zhu, Y. Lu, S. Gopal, J. Kahn, M. Kula, J. Pitman, R. Shah, E. Taropa, M. A. Merey, M. Baeuml, Z. Chen, L. E. Shafey, Y. Zhang, O. Sercinoglu, G. Tucker, E. Piqueras, M. Krikun, I. Barr, N. Savinov, I. Danihelka, B. Roelofs, A. White, A. Andreassen, T. von Glehn, L. Yagati, M. Kazemi, L. Gonzalez, M. Khalman, J. Sygnowski, A. Frechette, C. Smith, L. Culp, L. Proleev, Y. Luan, X. Chen, J. Lottes, N. Schucher, F. Lebron, A. Rrustemi, N. Clay, P. Crone, T. Kocisky, J. Zhao, B. Perz, D. Yu, H. Howard, A. Bloniarz, J. W. Rae, H. Lu, L. Sifre, M. Maggioni, F. Alcober, D. Garrette, M. Barnes, S. Thakoor, J. Austin, G. Barth-Maron, W. Wong, R. Joshi, R. Chaabouni, D. Fatiha, A. Ahuja, G. S. Tomar, E. Senter, M. Chadwick, I. Kornakov, N. Attaluri, I. Iturrate, R. Liu, Y. Li, S. Cogan, J. Chen, C. Jia, C. Gu, Q. Zhang, J. Grimstad, A. J. Hartman, X. Garcia, T. S. Pillai, J. Devlin, M. Laskin, D. de Las Casas, D. Valter, C. Tao, L. Blanco, A. P. Badia, D. Reitter, M. Chen, J. Brennan, C. Rivera, S. Brin, S. Iqbal, G. Surita, J. Labanowski, A. Rao, S. Winkler, E. Parisotto, Y. Gu, K. Olszewska, R. Addanki, A. Miech, A. Louis, D. Teplyashin, G. Brown, E. Catt, J. Balaguer, J. Xiang, P. Wang, Z. Ashwood, A. Briukhov, A. Webson, S. Ganapathy, S. Sanghavi, A. Kannan, M.-W. Chang, A. Stjerngren, J. Djolonga, Y. Sun, A. Bapna, M. Aitchison, P. Pejman, H. Michalewski, T. Yu, C. Wang, J. Love, J. Ahn, D. Bloxwich, K. Han, P. Humphreys, T. Sellam, J. Bradbury, V. Godbole, S. Samangooei, B. Damoc, A. Kaskasoli, S. M. R. Arnold, V. Vasudevan, S. Agrawal, J. Riesa, D. Lepikhin, R. Tanburn, S. Srinivasan, H. Lim, S. Hodkinson, P. Shyam, J. Ferret, S. Hand, A. Garg, T. L. Paine, J. Li, Y. Li, M. Giang, A. Neitz, Z. Abbas, S. York, M. Reid, E. Cole, A. Chowdhery, D. Das, D. RogoziÅĎska, V. Nikolaev, P. Sprechmann, Z. Nado, L. Zilka, F. Prost, L. He, M. Monteiro, G. Mishra, C. Welty, J. Newlan, D. Jia, M. Allamanis, C. H. Hu, R. de Liedekerke, J. Gilmer, C. Saroufim, S. Rijhwani, S. Hou, D. Shrivastava, A. Baddepudi, A. Goldin, A. Ozturel, A. Cassirer, Y. Xu, D. Sohn, D. Sachan, R. K. Amplayo, C. Swanson, D. Petrova, S. Narayan, A. Guez, S. Brahma, J. Landon, M. Patel, R. Zhao, K. Villela, L. Wang, W. Jia, M. Rahtz, M. GimÃĺnez, L. Yeung, J. Keeling, P. Georgiev, D. Mincu, B. Wu, S. Haykal, R. Saputro, K. Vodrahalli, J. Qin, Z. Cankara, A. Sharma, N. Fernando, W. Hawkins, B. Neyshabur, S. Kim, A. Hutter, P. Agrawal, A. Castro-Ros, G. van den Driessche, T. Wang, F. Yang, S. yiin Chang, P. Komarek, R. McIlroy, M. LuÄŅiÄĞ, G. Zhang, W. Farhan, M. Sharman, P. Natsev, P. Michel, Y. Bansal, S. Qiao, K. Cao, S. Shakeri, C. Butterfield, J. Chung, P. K. Rubenstein, S. Agrawal, A. Mensch, K. Soparkar, K. Lenc, T. Chung, A. Pope, L. Maggiore, J. Kay, P. Jhakra, S. Wang, J. Maynez, M. Phuong, T. Tobin, A. Tacchetti, M. Trebacz, K. Robinson, Y. Katariya, S. Riedel, P. Bailey, K. Xiao, N. Ghelani, L. Aroyo, A. Slone, N. Houlsby, X. Xiong, Z. Yang, E. Gribovskaya, J. Adler, M. Wirth, L. Lee, M. Li, T. Kagohara, J. Pavagadhi, S. Bridgers, A. Bortsova, S. Ghemawat, Z. Ahmed, T. Liu, R. Powell, V. Bolina, M. Iinuma, P. Zablotskaia, J. Besley, D.-W. Chung, T. Dozat, R. Comanescu, X. Si, J. Greer, G. Su, M. Polacek, R. L. Kaufman, S. Tokumine, H. Hu, E. Buchatskaya, Y. Miao, M. Elhawaty, A. Siddhant, N. Tomasev, J. Xing, C. Greer, H. Miller, S. Ashraf, A. Roy, Z. Zhang, A. Ma, A. Filos, M. Besta, R. Blevins, T. Klimenko, C.-K. Yeh, S. Changpinyo, J. Mu, O. Chang, M. Pajarskas, C. Muir, V. Cohen, C. L. Lan, K. Haridasan, A. Marathe, S. Hansen, S. Douglas, R. Samuel, M. Wang, S. Austin, C. Lan, J. Jiang, J. Chiu, J. A. Lorenzo, L. L. SjÃűsund, S. Cevey, Z. Gleicher, T. Avrahami, A. Boral, H. Srinivasan, V. Selo, R. May, K. Aisopos, L. Hussenot, L. B. Soares, K. Baumli, M. B. Chang, A. Recasens, B. Caine, A. Pritzel, F. Pavetic, F. Pardo, A. Gergely, J. Frye, V. Ramasesh, D. Horgan, K. Badola, N. Kassner, S. Roy, E. Dyer, V. C. Campos, A. Tomala, Y. Tang, D. E. Badawy, E. White, B. Mustafa, O. Lang, A. Jindal, S. Vikram, Z. Gong, S. Caelles, R. Hemsley, G. Thornton, F. Feng, W. Stokowiec, C. Zheng, P. Thacker, ÃĞaÄ§lar ÃIJnlÃij, Z. Zhang, M. Saleh, J. Svensson, M. Bileschi, P. Patil, A. Anand, R. Ring, K. Tsihlas, A. Vezer, M. Selvi, T. Shevlane, M. Rodriguez, T. Kwiatkowski, S. Daruki, K. Rong, A. Dafoe, N. FitzGerald, K. Gu-Lemberg, M. Khan, L. A. Hendricks, M. Pellat, V. Feinberg, J. Cobon-

Kerr, T. Sainath, M. Rauh, S. H. Hashemi, R. Ives, Y. Hasson, E. Noland, Y. Cao, N. Byrd, L. Hou, Q. Wang, T. Sottiaux, M. Paganini, J.-B. Lespiau, A. Moufarek, S. Hassan, K. Shivakumar, J. van Amersfoort, A. Mandhane, P. Joshi, A. Goyal, M. Tung, A. Brock, H. Sheahan, V. Misra, C. Li, N. RakiÄĞeviÄĞ, M. Dehghani, F. Liu, S. Mittal, J. Oh, S. Noury, E. Sezener, F. Huot, M. Lamm, N. D. Cao, C. Chen, S. Mudgal, R. Stella, K. Brooks, G. Vasudevan, C. Liu, M. Chain, N. Melinkeri, A. Cohen, V. Wang, K. Seymore, S. Zubkov, R. Goel, S. Yue, S. Krishnakumaran, B. Albert, N. Hurley, M. Sano, A. Mohananey, J. Joughin, E. Filonov, T. KÄŹpa, Y. Eldawy, J. Lim, R. Rishi, S. Badiezadegan, T. Bos, J. Chang, S. Jain, S. G. S. Padmanabhan, S. Puttagunta, K. Krishna, L. Baker, N. Kalb, V. Bedapudi, A. Kurzrok, S. Lei, A. Yu, O. Litvin, X. Zhou, Z. Wu, S. Sobell, A. Siciliano, A. Papir, R. Neale, J. Bragagnolo, T. Toor, T. Chen, V. Anklin, F. Wang, R. Feng, M. Gholami, K. Ling, L. Liu, J. Walter, H. Moghaddam, A. Kishore, J. Adamek, T. Mercado, J. Mallinson, S. Wandekar, S. Cagle, E. Ofek, G. Garrido, C. Lombriser, M. Mukha, B. Sun, H. R. Mohammad, J. Matak, Y. Qian, V. Peswani, P. Janus, Q. Yuan, L. Schelin, O. David, A. Garg, Y. He, O. Duzhyi, A. ÃĎlgmyr, T. Lottaz, Q. Li, V. Yadav, L. Xu, A. Chinien, R. Shivanna, A. Chuklin, J. Li, C. Spadine, T. Wolfe, K. Mohamed, S. Das, Z. Dai, K. He, D. von Dincklage, S. Upadhyay, A. Maurya, L. Chi, S. Krause, K. Salama, P. G. Rabinovitch, P. K. R. M, A. Selvan, M. Dektiarev, G. Ghiasi, E. Guven, H. Gupta, B. Liu, D. Sharma, I. H. Shtacher, S. Paul, O. Akerlund, F.-X. Aubet, T. Huang, C. Zhu, E. Zhu, E. Teixeira, M. Fritze, F. Bertolini, L.-E. Marinescu, M. BÃűlle, D. Paulus, K. Gupta, T. Latkar, M. Chang, J. Sanders, R. Wilson, X. Wu, Y.-X. Tan, L. N. Thiet, T. Doshi, S. Lall, S. Mishra, W. Chen, T. Luong, S. Benjamin, J. Lee, E. Andrejczuk, D. Rabiej, V. Ranjan, K. Styrc, P. Yin, J. Simon, M. R. Harriott, M. Bansal, A. Robsky, G. Bacon, D. Greene, D. Mirylenka, C. Zhou, O. Sarvana, A. Goyal, S. Andermatt, P. Siegler, B. Horn, A. Israel, F. Pongetti, C.-W. L. Chen, M. Selvatici, P. Silva, K. Wang, J. Tolins, K. Guu, R. Yogev, X. Cai, A. Agostini, M. Shah, H. Nguyen, N. Ã. Donnaile, S. Pereira, L. Friso, A. Stambler, A. Kurzrok, C. Kuang, Y. Romanikhin, M. Geller, Z. Yan, K. Jang, C.-C. Lee, W. Fica, E. Malmi, Q. Tan, D. Banica, D. Balle, R. Pham, Y. Huang, D. Avram, H. Shi, J. Singh, C. Hidey, N. Ahuja, P. Saxena, D. Dooley, S. P. Potharaju, E. O'Neill, A. Gokulchandran, R. Foley, K. Zhao, M. Dusenberry, Y. Liu, P. Mehta, R. Kotikalapudi, C. Safranek-Shrader, A. Goodman, J. Kessinger, E. Globen, P. Kolhar, C. Gorgolewski, A. Ibrahim, Y. Song, A. Eichenbaum, T. Brovelli, S. Potluri, P. Lahoti, C. Baetu, A. Ghorbani, C. Chen, A. Crawford, S. Pal, M. Sridhar, P. Gurita, A. Mujika, I. Petrovski, P.-L. Cedoz, C. Li, S. Chen, N. D. Santo, S. Goyal, J. Punjabi, K. Kappaganthu, C. Kwak, P. LV, S. Velury, H. Choudhury, J. Hall, P. Shah, R. Figueira, M. Thomas, M. Lu, T. Zhou, C. Kumar, T. Jurdi, S. Chikkerur, Y. Ma, A. Yu, S. Kwak, V. ÃĎhdel, S. Rajayogam, T. Choma, F. Liu, A. Barua, C. Ji, J. H. Park, V. Hellendoorn, A. Bailey, T. Bilal, H. Zhou, M. Khatir, C. Sutton, W. Rzadkowski, F. Macintosh, K. Shagin, P. Medina, C. Liang, J. Zhou, P. Shah, Y. Bi, A. Dankovics, S. Banga, S. Lehmann, M. Bredesen, Z. Lin, J. E. Hoffmann, J. Lai, R. Chung, K. Yang, N. Balani, A. BraÅ¿inskas, A. Sozanschi, M. Hayes, H. F. Alcalde, P. Makarov, W. Chen, A. Stella, L. Snijders, M. Mandl, A. KÃďrrman, P. Nowak, X. Wu, A. Dyck, K. Vaidyanathan, R. R, J. Mallet, M. Rudominer, E. Johnston, S. Mittal, A. Udathu, J. Christensen, V. Verma, Z. Irving, A. Santucci, G. Elsayed, E. Davoodi, M. Georgiev, I. Tenney, N. Hua, G. Cideron, E. Leurent, M. Alnahlawi, I. Georgescu, N. Wei, I. Zheng, D. Scandinaro, H. Jiang, J. Snoek, M. Sundararajan, X. Wang, Z. Ontiveros, I. Karo, J. Cole, V. Rajashekhar, L. Tumeh, E. Ben-David, R. Jain, J. Uesato, R. Datta, O. Bunyan, S. Wu, J. Zhang, P. Stanczyk, Y. Zhang, D. Steiner, S. Naskar, M. Azzam, M. Johnson, A. Paszke, C.-C. Chiu, J. S. Elias, A. Mohiuddin, F. Muhammad, J. Miao, A. Lee, N. Vieillard, J. Park, J. Zhang, J. Stanway, D. Garmon, A. Karmarkar, Z. Dong, J. Lee, A. Kumar, L. Zhou, J. Evens, W. Isaac, G. Irving, E. Loper, M. Fink, I. Arkatkar, N. Chen, I. Shafran, I. Petrychenko, Z. Chen, J. Jia, A. Levskaya, Z. Zhu, P. Grabowski, Y. Mao, A. Magni, K. Yao, J. Snaider, N. Casagrande, E. Palmer, P. Suganthan,

A. CastaÃśo, I. Giannoumis, W. Kim, M. RybiÅĎski, A. Sreevatsa, J. Prendki, D. Soergel, A. Goedeckemeyer, W. Gierke, M. Jafari, M. Gaba, J. Wiesner, D. G. Wright, Y. Wei, H. Vashisht, Y. Kulizhskaya, J. Hoover, M. Le, L. Li, C. Iwuanyanwu, L. Liu, K. Ramirez, A. Khorlin, A. Cui, T. LIN, M. Wu, R. Aguilar, K. Pallo, A. Chakladar, G. Perng, E. A. Abellan, M. Zhang, I. Dasgupta, N. Kushman, I. Penchev, A. Repina, X. Wu, T. van der Weide, P. Ponnapalli, C. Kaplan, J. Simsa, S. Li, O. Dousse, F. Yang, J. Piper, N. Ie, R. Pasumarthi, N. Lintz, A. Vijayakumar, D. Andor, P. Valenzuela, M. Lui, C. Paduraru, D. Peng, K. Lee, S. Zhang, S. Greene, D. D. Nguyen, P. Kurylowicz, C. Hardin, L. Dixon, L. Janzer, K. Choo, Z. Feng, B. Zhang, A. Singhal, D. Du, D. McKinnon, N. Antropova, T. Bolukbasi, O. Keller, D. Reid, D. Finchelstein, M. A. Raad, R. Crocker, P. Hawkins, R. Dadashi, C. Gaffney, K. Franko, A. Bulanova, R. Leblond, S. Chung, H. Askham, L. C. Cobo, K. Xu, F. Fischer, J. Xu, C. Sorokin, C. Alberti, C.-C. Lin, C. Evans, A. Dimitriev, H. Forbes, D. Banarse, Z. Tung, M. Omernick, C. Bishop, R. Sterneck, R. Jain, J. Xia, E. Amid, F. Piccinno, X. Wang, P. Banzal, D. J. Mankowitz, A. Polozov, V. Krakovna, S. Brown, M. Bateni, D. Duan, V. Firoiu, M. Thotakuri, T. Natan, M. Geist, S. tan Girgin, H. Li, J. Ye, O. Roval, R. Tojo, M. Kwong, J. Lee-Thorp, C. Yew, D. Sinopalnikov, S. Ramos, J. Mellor, A. Sharma, K. Wu, D. Miller, N. Sonnerat, D. Vnukov, R. Greig, J. Beattie, E. Caveness, L. Bai, J. Eisenschlos, A. Korchemniy, T. Tsai, M. Jasarevic, W. Kong, P. Dao, Z. Zheng, F. Liu, F. Yang, R. Zhu, T. H. Teh, J. Sanmiya, E. Gladchenko, N. Trdin, D. Toyama, E. Rosen, S. Tavakkol, L. Xue, C. Elkind, O. Woodman, J. Carpenter, G. Papamakarios, R. Kemp, S. Kafle, T. Grunina, R. Sinha, A. Talbert, D. Wu, D. Owusu-Afriyie, C. Du, C. Thornton, J. Pont-Tuset, P. Narayana, J. Li, S. Fatehi, J. Wieting, O. Ajmeri, B. Uria, Y. Ko, L. Knight, A. HÃľliou, N. Niu, S. Gu, C. Pang, Y. Li, N. Levine, A. Stolovich, R. Santamaria-Fernandez, S. Goenka, W. Yustalim, R. Strudel, A. Elqursh, C. Deck, H. Lee, Z. Li, K. Levin, R. Hoffmann, D. Holtmann-Rice, O. Bachem, S. Arora, C. Koh, S. H. Yeganeh, S. PÃţder, M. Tariq, Y. Sun, L. Ionita, M. Seyedhosseini, P. Tafti, Z. Liu, A. Gulati, J. Liu, X. Ye, B. Chrzaszcz, L. Wang, N. Sethi, T. Li, B. Brown, S. Singh, W. Fan, A. Parisi, J. Stanton, V. Koverkathu, C. A. Choquette-Choo, Y. Li, T. Lu, A. Ittycheriah, P. Shroff, M. Varadarajan, S. Bahargam, R. Willoughby, D. Gaddy, G. Desjardins, M. Cornero, B. Robenek, B. Mittal, B. Albrecht, A. Shenoy, F. Moiseev, H. Jacobsson, A. Ghaffarkhah, M. RiviÃĺre, A. Walton, C. Crepy, A. Parrish, Z. Zhou, C. Farabet, C. Radebaugh, P. Srinivasan, C. van der Salm, A. Fidjeland, S. Scellato, E. Latorre-Chimoto, H. Klimczak-PluciÅĎska, D. Bridson, D. de Cesare, T. Hudson, P. Mendolicchio, L. Walker, A. Morris, M. Mauger, A. Guseynov, A. Reid, S. Odoom, L. Loher, V. Cotruta, M. Yenugula, D. Grewe, A. Petrushkina, T. Duerig, A. Sanchez, S. Yadlowsky, A. Shen, A. Globerson, L. Webb, S. Dua, D. Li, S. Bhupatiraju, D. Hurt, H. Qureshi, A. Agarwal, T. Shani, M. Eyal, A. Khare, S. R. Belle, L. Wang, C. Tekur, M. S. Kale, J. Wei, R. Sang, B. Saeta, T. Liechty, Y. Sun, Y. Zhao, S. Lee, P. Nayak, D. Fritz, M. R. Vuyyuru, J. Aslanides, N. Vyas, M. Wicke, X. Ma, E. Eltyshev, N. Martin, H. Cate, J. Manyika, K. Amiri, Y. Kim, X. Xiong, K. Kang, F. Luisier, N. Tripuraneni, D. Madras, M. Guo, A. Waters, O. Wang, J. Ainslie, J. Baldridge, H. Zhang, G. Pruthi, J. Bauer, F. Yang, R. Mansour, J. Gelman, Y. Xu, G. Polovets, J. Liu, H. Cai, W. Chen, X. Sheng, E. Xue, S. Ozair, C. Angermueller, X. Li, A. Sinha, W. Wang, J. Wiesinger, E. Koukoumidis, Y. Tian, A. Iyer, M. Gurumurthy, M. Goldenson, P. Shah, M. Blake, H. Yu, A. Urbanowicz, J. Palomaki, C. Fernando, K. Durden, H. Mehta, N. Momchev, E. Rahimtoroghi, M. Georgaki, A. Raul, S. Ruder, M. Redshaw, J. Lee, D. Zhou, K. Jalan, D. Li, B. Hechtman, P. Schuh, M. Nasr, K. Milan, V. Mikulik, J. Franco, T. Green, N. Nguyen, J. Kelley, A. Mahendru, A. Hu, J. Howland, B. Vargas, J. Hui, K. Bansal, V. Rao, R. Ghiya, E. Wang, K. Ye, J. M. Sarr, M. M. Preston, M. Elish, S. Li, A. Kaku, J. Gupta, I. Pasupat, D.-C. Juan, M. Someswar, T. M., X. Chen, A. Amini, A. Fabrikant, E. Chu, X. Dong, A. Muthal, S. Buthpitiya, S. Jauhari, N. Hua, U. Khandelwal, A. Hitron, J. Ren, L. Rinaldi, S. Drath, A. Dabush, N.-J. Jiang, H. Godhia, U. Sachs,

A. Chen, Y. Fan, H. Taitelbaum, H. Noga, Z. Dai, J. Wang, C. Liang, J. Hamer, C.-S. Ferng, C. Elkind, A. Atias, P. Lee, V. ListÃŋk, M. Carlen, J. van de Kerkhof, M. Pikus, K. Zaher, P. MÃijller, S. Zykova, R. Stefanec, V. Gatsko, C. Hirnschall, A. Sethi, X. F. Xu, C. Ahuja, B. Tsai, A. Stefanoiu, B. Feng, K. Dhandhania, M. Katyal, A. Gupta, A. Parulekar, D. Pitta, J. Zhao, V. Bhatia, Y. Bhavnani, O. Alhadlaq, X. Li, P. Danenberg, D. Tu, A. Pine, V. Filippova, A. Ghosh, B. Limonchik, B. Urala, C. K. Lanka, D. Clive, Y. Sun, E. Li, H. Wu, K. Hongtongsak, I. Li, K. Thakkar, K. Omarov, K. Majmundar, M. Alverson, M. Kucharski, M. Patel, M. Jain, M. Zabelin, P. Pelagatti, R. Kohli, S. Kumar, J. Kim, S. Sankar, V. Shah, L. Ramachandruni, X. Zeng, B. Bariach, L. Weidinger, T. Vu, A. Andreev, A. He, K. Hui, S. Kashem, A. Subramanya, S. Hsiao, D. Hassabis, K. Kavukcuoglu, A. Sadovsky, Q. Le, T. Strohman, Y. Wu, S. Petrov, J. Dean, and O. Vinyals, "Gemini: A family of highly capable multimodal models," 2024.

[61] L.-W. Chang, W. Bao, Q. Hou, C. Jiang, N. Zheng, Y. Zhong, X. Zhang, Z. Song, Z. Jiang, H. Lin, X. Jin, and X. Liu, "Flux: Fast software-based communication overlap on gpus through kernel fusion," 2024.

[62] C. Team, "Cursor: An ai-powered coding assistant." https://github.com/getcursor/cursor, 2023. GitHub repository.

[63] GitHub and OpenAI, "Github copilot: Your ai pair programmer." https://github.com/features/copilot, 2023. Software tool.

[64] X. Wang, B. Li, Y. Song, F. F. Xu, X. Tang, M. Zhuge, J. Pan, Y. Song, B. Li, J. Singh, H. H. Tran, F. Li, R. Ma, M. Zheng, B. Qian, Y. Shao, N. Muennighoff, Y. Zhang, B. Hui, J. Lin, R. Brennan, H. Peng, H. Ji, and G. Neubig, "Opendevin: An open platform for ai software developers as generalist agents," 2024.

[65] G. Team, "Geospy.ai: Geospatial intelligence platform." https://geospy.ai, 2023.

[66] J. Ho, A. Jain, and P. Abbeel, "Denoising diffusion probabilistic models," 2020.

[67] R. Rombach, A. Blattmann, D. Lorenz, P. Esser, and B. Ommer, "High-resolution image synthesis with latent diffusion models," 2022.

[68] Automatic1111, "Stable diffusion webui." https://github.com/AUTOMATIC1111/stable-diffusion-webui, 2023. GitHub repository.

[69] C. Team, "Comfyui: A powerful and modular stable diffusion graph node system." https://github.com/comfyanonymous/ComfyUI, 2023. GitHub repository.

[70] C. Lu, Y. Zhou, F. Bao, J. Chen, C. Li, and J. Zhu, "Dpm-solver++: Fast solver for guided sampling of diffusion probabilistic models," 2023.

[71] G. Research, "Colaboratory: Collaborative data science." https://colab.research.google.com/, 2023.

[72] C. Team, "Civitai: Ai model sharing platform." https://civitai.com, 2023.