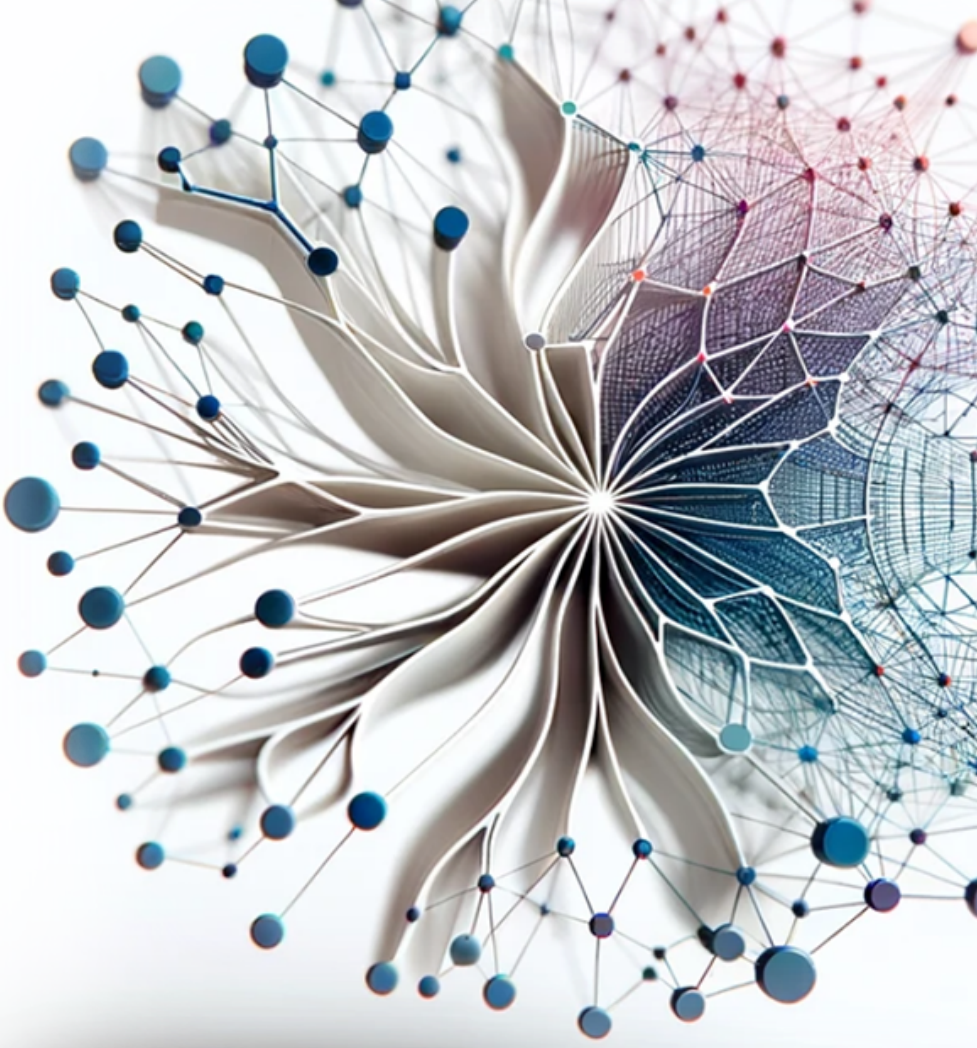


Machine Learning Systems

Vijay
Janapa Reddi

Machine Learning Systems

Principles and Practices of Engineering Artificially Intelligent Systems

Prof. Vijay Janapa Reddi
School of Engineering and Applied Sciences
Harvard University

With heartfelt gratitude to the community for their invaluable contributions and steadfast support.

Table of contents

Preface

Welcome to Machine Learning Systems, your gateway to the fast-paced world of machine learning (ML) systems. This book is an extension of the CS249r course at Harvard University, taught by Prof. Vijay Janapa Reddi, and is the result of a collaborative effort involving students, professionals, and the broader community of AI practitioners.

We've created this open-source book to demystify the process of building efficient and scalable ML systems. Our goal is to provide a comprehensive guide that covers the principles, practices, and challenges of developing robust ML pipelines for deployment. This isn't a static textbook—it's a living, evolving resource designed to keep pace with advancements in the field.

> "If you want to go fast, go alone. If you want to go far, go together."
> – African Proverb

As a living and breathing resource, this book is a continual work in progress, reflecting the ever-evolving nature of machine learning systems. Advancements in the ML landscape drive our commitment to keeping this resource updated with the latest insights, techniques, and best practices. We warmly invite you to join us on this journey by contributing your expertise, feedback, and ideas.

Global Outreach

Thank you to all our readers and visitors. Your engagement with the material keeps us motivated.

Why We Wrote This Book

While there are plenty of resources that focus on the algorithmic side of machine learning, resources on the systems side of things are few and far between. This gap inspired us to create this book—a resource dedicated to the principles and practices of building efficient and scalable ML systems.

Our vision for this book and its broader mission is deeply rooted in the transformative potential of AI and the need to make AI education globally accessible to all. To learn more about the inspiration behind this project and the values driving its creation, we encourage you to read the Author's Note.

Want to Help Out?

This is a collaborative project, and your input matters! If you'd like to contribute, check out our contribution guidelines. Feedback, corrections, and new ideas are welcome—simply file a GitHub issue.

What's Next?

If you're ready to dive deeper into the book's structure, learning objectives, and practical use, visit the About the Book section for more details.

Author's Note

AI is bound to transform the world in profound ways, much like computers and the Internet revolutionized every aspect of society in the 20th century. From systems that generate creative content to those driving breakthroughs in drug discovery, AI is ushering in a new era—one that promises to be even more transformative in its scope and impact. But how do we make it accessible to everyone?

With its transformative power comes an equally great responsibility for those who access it or work with it. Just as we expect companies to wield their influence ethically, those of us in academia bear a parallel responsibility: to share our knowledge openly, so it benefits everyone—not just a select few. This conviction inspired the creation of this book—an open-source resource aimed at making AI education, particularly in AI engineering and systems, inclusive and accessible to everyone from all walks of life.

My passion for creating, curating, and editing this content has been deeply influenced by landmark textbooks that have profoundly shaped both my academic and personal journey. Whether I studied them cover to cover or drew insights from key passages, these resources fundamentally shaped the way I think. I reflect on the books that guided my path: works by Turing Award winners such as David Patterson and John Hennessy—pioneers in computer architecture and system design—and foundational research papers by luminaries like Yann LeCun, Geoffrey Hinton, and Yoshua Bengio. In some small part, my hope is that this book will inspire students to chart their own unique paths.

I am optimistic about what lies ahead for AI. It has the potential to solve global challenges and unlock creativity in ways we have yet to imagine. To achieve this, however, we must train the next generation of AI engineers and practitioners—those who can transform novel AI algorithms into working systems that enable real-world application. This book is a step toward curating the material needed to build the next generation of AI engineers who will transform today's visions into tomorrow's reality.

This book is a work in progress, but knowing that even one learner benefits from its content motivates me to continually refine and expand it. To that end, if there's one thing I ask of readers, it's this: please show your support by starring the GitHub repository here. Your star reflects your belief in this mission—not just to me, but to the growing global community of learners, ed-

ucators, and practitioners. This small act is more than symbolic—it amplifies the importance of making AI education accessible.

I am a student of my own writing, and every chapter of this book has taught me something new—thanks to the numerous people who have played, and continue to play, an important role in shaping this work. Professors, students, practitioners, and researchers contributed by offering suggestions, sharing expertise, identifying errors, and proposing improvements. Every interaction, whether a detailed critique or a simple correction from a GitHub contributor, has been a lesson in itself. These contributions have not only refined the material but also deepened my understanding of how knowledge grows through collaboration. This book is, therefore, not solely my work; it is a shared endeavor, reflecting the collective spirit of those dedicated to sharing their knowledge and effort.

This book is dedicated to the loving memory of my father. His passion for education, endless curiosity, generosity in sharing knowledge, and unwavering commitment to quality challenge me daily to strive for excellence in all I do. In his honor, I extend this dedication to teachers and mentors everywhere, whose efforts and guidance transform lives every day. Your selfless contributions remind me to persevere.

Last but certainly not least, this work would not be possible without the unwavering support of my wonderful wife and children. Their love, patience, and encouragement form the foundation that enables me to pursue my passion and bring this work to life. For this, and so much more, I am deeply grateful.

— Prof. Vijay Janapa Reddi

About the Book

Overview

Purpose of the Book

Welcome to this collaborative textbook. It originated as part of the *CS249r: Tiny Machine Learning* course that Prof. Vijay Janapa Reddi teaches at Harvard University.

The goal of this book is to provide a resource for educators and learners seeking to understand the principles and practices of machine learning systems. This book is continually updated to incorporate the latest insights and effective teaching strategies with the intent that it remains a valuable resource in this fast-evolving field. So please check back often!

Context and Development

The book reflects a blend of pedagogical expertise and cutting-edge research. Developed collaboratively with contributions from students, researchers, and practitioners, it bridges academic rigor and real-world application.

We've designed the book to evolve alongside advancements in the field, fostering a collaborative environment where knowledge can grow and adapt.

What to Expect

This textbook explores the foundational principles, practical workflows, and critical challenges of building and deploying machine learning systems. Starting with **foundational concepts**, it progresses through **engineering principles**, examines **operational considerations** for deploying AI systems, and concludes by reflecting on the societal and technological implications of machine learning.

Learning Goals

Key Learning Outcomes

This book is structured with Bloom's Taxonomy in mind, which defines six levels of learning, ranging from foundational knowledge to advanced creative thinking:

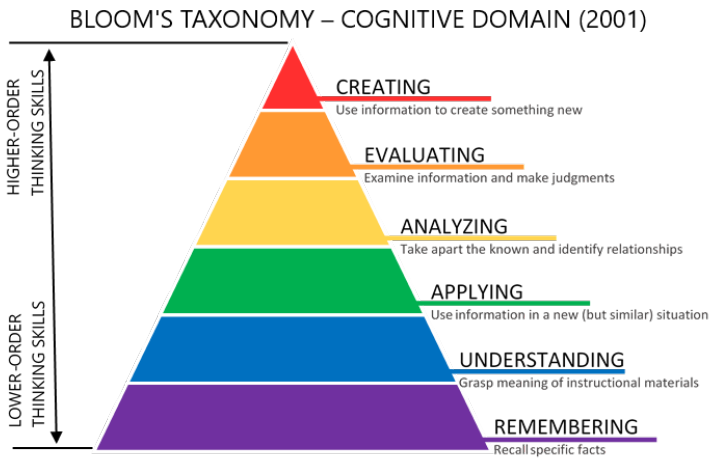


Figure 5.1: Bloom's Taxonomy (2021 edition).

1. **Remembering**: Recalling basic facts and concepts.
2. **Understanding**: Explaining ideas or processes.
3. **Applying**: Using knowledge in new situations.
4. **Analyzing**: Breaking down information into components.
5. **Evaluating**: Making judgments based on criteria and standards.
6. **Creating**: Producing original work or solutions.

Learning Objectives

This book supports readers in:

1. **Understanding Fundamentals**: Explain the foundational principles of machine learning, including theoretical underpinnings and practical applications.
2. **Analyzing System Components**: Evaluate the critical components of AI systems and their roles within various architectures.
3. **Designing Workflows**: Outline workflows for developing machine learning systems, from data collection to deployment.
4. **Optimizing Models**: Apply methods to enhance performance, such as hyperparameter tuning and regularization.
5. **Evaluating Ethical Implications**: Analyze societal impacts and address potential biases in AI systems.

6. **Exploring Applications**: Investigate real-world use cases across diverse domains.
7. **Considering Deployment Challenges**: Address security, scalability, and maintainability in real-world systems.
8. **Envisioning Future Trends**: Reflect on emerging challenges and technologies in machine learning.

AI Learning Companion

Throughout this resource, you'll find **SocratiQ**—an AI learning assistant designed to enhance your learning experience. Inspired by the Socratic method of teaching, SocratiQ combines interactive quizzes, personalized assistance, and real-time feedback to help you reinforce your understanding and create new connections. As part of our experiment with Generative AI technologies, SocratiQ encourages critical thinking and active engagement with the material.

SocratiQ is still a work in progress, and we welcome your feedback to make it better. For more details about how SocratiQ works and how to get the most out of it, visit the AI Learning Companion page.

How to Navigate This Book

Book Structure

The book is organized into four main parts, each building on the previous one:

1. **The Essentials (Chapters 1-4)**
   Core principles, components, and architectures that underpin machine learning systems.
2. **Engineering Principles (Chapters 5-13)**
   Covers workflows, data engineering, optimization strategies, and operational challenges in system design.
3. **AI Best Practice (Chapters 14-18)**
   Focuses on key considerations for deploying AI systems in real-world environments, including security, privacy, robustness, and sustainability.
4. **Closing Perspectives (Chapter 19-20)**
   Synthesizes key lessons and explores emerging trends shaping the future of ML systems.

Suggested Reading Paths

- **Beginners**: Start with *The Essentials* to build a strong conceptual base before progressing to other parts.
- **Practitioners**: Focus on *Engineering Principles* and *AI in Practice* for hands-on, real-world insights.
- **Researchers**: Dive into *AI in Practice* and *Closing Perspectives* to explore advanced topics and societal implications.

Modular Design

The book is modular, allowing readers to explore chapters independently or sequentially. Each chapter includes supplementary resources:

- **Slides** summarizing key concepts.
- **Videos** providing in-depth explanations.
- **Exercises** reinforcing understanding.
- **Labs** offering practical, hands-on experience.

While several of these resources are still a work in progress, we believe it's better to share valuable insights and tools as they become available rather than wait for everything to be perfect. After all, progress is far more important than perfection, and your feedback will help us improve and refine this resource over time.

Additionally, we try to reuse and build upon the incredible work created by amazing experts in the field, rather than reinventing everything from scratch. This philosophy reflects the fundamental essence of community-driven learning: collaboration, sharing knowledge, and collectively advancing our understanding.

Transparency and Collaboration

This book is a community-driven project, with content generated collaboratively by numerous contributors over time. The content creation process may have involved various editing tools, including generative AI technology. As the main author, editor, and curator, Prof. Vijay Janapa Reddi maintains human oversight to ensure the content is accurate and relevant.

However, no one is perfect, and inaccuracies may still exist. Your feedback is highly valued, and we encourage you to provide corrections or suggestions. This collaborative approach is crucial for maintaining high-quality information and making it globally accessible.

Copyright and Licensing

Join the Community

This textbook is more than just a resource—it's an invitation to collaborate and learn together. Engage in community discussions to share insights, tackle challenges, and learn alongside fellow students, researchers, and practitioners.

Whether you're a student starting your journey, a practitioner solving real-world challenges, or a researcher exploring advanced concepts, your contributions will enrich this learning community. Introduce yourself, share your goals, and let's collectively build a deeper understanding of machine learning systems.

#

Engineering Principles

Chapter 6

Data Engineering

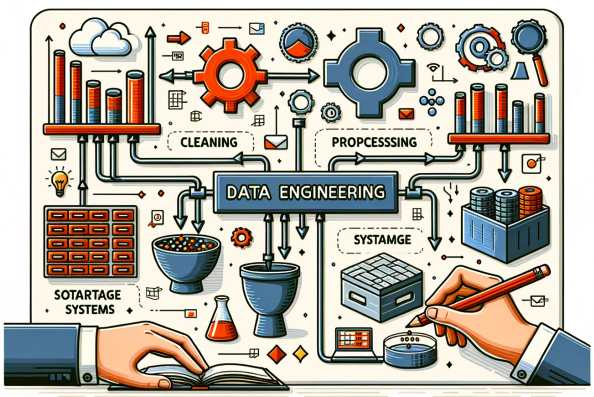


Figure 6.1: *DALL·E 3 Prompt: Create a rectangular illustration visualizing the concept of data engineering. Include elements such as raw data sources, data processing pipelines, storage systems, and refined datasets. Show how raw data is transformed through cleaning, processing, and storage to become valuable information that can be analyzed and used for decision-making.*

Purpose

How does data shape ML systems engineering?

Machine learning is generally often overshadowed by the allure of sophisticated algorithms, when in fact data plays a foundational role in determining an AI system's capabilities and limitations. We need to undersatnd the core principles of data in ML systems, exploring how the acquisition, processing, storage, and governance of data directly impact the performance, reliability, and ethical considerations of AI systems. By understanding these fundamental concepts, we can unlock the true potential of AI and build a solid foundation of high-quality ML solutions.

Learning Objectives

- Analyze different data sourcing methods (datasets, web scraping, crowdsourcing, synthetic data).
- Explain the importance of data labeling and ensure label quality.
- Evaluate data storage systems for ML workloads (databases, data warehouses, data lakes).
- Describe the role of data pipelines in ML systems.
- Explain the importance of data governance in ML (security, privacy, ethics).
- Identify key challenges in data engineering for ML.

6.1 Overview

Data is the foundation of modern machine learning systems, governing their success through its quality and accessibility. Despite its pivotal role, data engineering is often overlooked compared to algorithm design and model development. However, the effectiveness of any machine learning system hinges on the robustness of its data pipeline. As machine learning applications become more sophisticated, the challenges associated with curating, cleaning, organizing, and storing data have grown significantly. These activities have emerged as some of the most resource-intensive aspects of the data engineering process, requiring sustained effort and attention.

The concept of "Data Cascades," introduced by Sambasivan et al. (2021), highlights the systemic failures that can arise when data quality issues are left unaddressed. Errors originating during data collection or processing stages can compound over time, creating cascading effects that lead to model failures, costly retraining, or even project termination. The failures of IBM Watson Health in 2019, where flawed training data resulted in unsafe and incorrect cancer treatment recommendations (Strickland 2019), show the real-world consequences of neglecting data quality and its associated engineering requirements.

It is therefore unsurprising that data scientists spend the majority of their time—up to 60%, as shown in Figure 6.2—is spent on cleaning and organizing data. This statistic highlights the critical need to prioritize data-related challenges early in the pipeline to avoid downstream issues and ensure the effectiveness of machine learning systems.

This chapter examines the lifecycle of data engineering in machine learning systems, presenting an overview of the stages involved and the unique challenges at each step. The discussion begins with the identification and sourcing of data, exploring diverse origins such as pre-existing datasets, web scraping, crowdsourcing, and synthetic data generation. Special attention is given to the complexities of integrating heterogeneous sources, validating incoming data, and handling errors during ingestion.

Next, the chapter explores the transformation of raw data into machine learning-ready formats. This process involves cleaning, normalizing, and

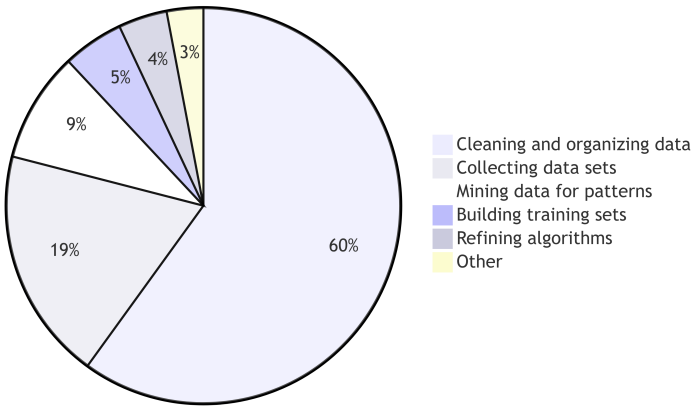


Figure 6.2: What do data scientists spend most of their time on?

extracting features, tasks that are critical to optimizing model learning and ensuring robust performance. The challenges of scale and computational efficiency are also discussed, as they are particularly important for systems that operate on vast and complex datasets.

Beyond data processing, the chapter addresses the intricacies of data labeling, a crucial step for supervised learning systems. Effective labeling requires sound annotation methodologies and advanced techniques such as AI-assisted annotation to ensure the accuracy and consistency of labeled data. Challenges such as bias and ambiguity in labeling are explored, with examples illustrating their potential impact on downstream tasks.

The chapter also examines the storage and organization of data, a vital aspect of supporting machine learning pipelines across their lifecycle. Topics such as storage system design, feature stores, caching strategies, and access patterns are discussed, with a focus on ensuring scalability and efficiency. Governance is highlighted as a key component of data storage and management, emphasizing the importance of compliance with privacy regulations, ethical considerations, and the use of documentation frameworks to maintain transparency and accountability.

This chapter provides an exploration of data engineering practices necessary for building and maintaining effective machine learning systems. The end goal is to emphasize the often-overlooked importance of data in enabling the success of machine learning applications.

6.2 Problem Definition

In many machine learning domains, sophisticated algorithms take center stage, while the fundamental importance of data quality is often overlooked. This neglect gives rise to "Data Cascades" by Sambasivan et al. (2021)—events where lapses in data quality compound, leading to negative downstream consequences such as flawed predictions, project terminations, and even potential harm to communities.

Figure 6.3 illustrates these potential data pitfalls at every stage and how they influence the entire process down the line. The influence of data collection er-

rors is especially pronounced. As depicted in the figure, any lapses in this initial stage will become apparent at later stages (in model evaluation and deployment) and might lead to costly consequences, such as abandoning the entire model and restarting anew. Therefore, investing in data engineering techniques from the onset will help us detect errors early, mitigating the cascading effects illustrated in the figure.

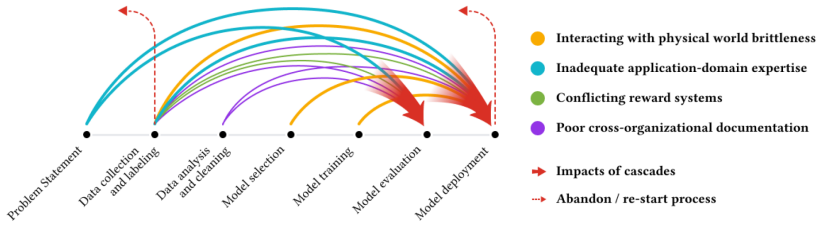


Figure 6.3: Data cascades: compounded costs. Source: Sambasivan et al. (2021).

Despite many ML professionals recognizing the importance of data, numerous practitioners report facing these cascades. This highlights a systemic issue: while the allure of developing advanced models remains, data often needs to be more appreciated.

This emphasis on data quality and proper problem definition is fundamental across all types of ML systems. As Sculley et al. (2021) emphasize, it is important to distinguish ML-specific problem framing from the broader context of general software development. Whether developing recommendation engines processing millions of user interactions, computer vision systems analyzing medical images, or natural language models handling diverse text data, each system brings unique challenges that must be carefully considered from the outset. Production ML systems are particularly sensitive to data quality issues, as they must handle continuous data streams, maintain consistent processing pipelines, and adapt to evolving patterns while maintaining performance standards.

A solid project foundation is essential for setting the trajectory and ensuring the eventual success of any initiative. At the heart of this foundation lies the crucial first step: identifying a clear problem to solve. This could involve challenges like developing a recommendation system that effectively handles cold-start scenarios, or creating a classification model that maintains consistent accuracy across diverse population segments.

As we will explore later in this chapter, establishing clear objectives provides a unified direction that guides the entire project. These objectives might include creating representative datasets that account for various real-world scenarios. Equally important is defining specific benchmarks, such as prediction accuracy and system latency, which offer measurable outcomes to gauge progress and success.

Throughout this process, engaging with stakeholders—from end-users to business leaders—provides invaluable insights that ensure the project remains aligned with real-world needs and expectations.

Generally, in ML, problem definition has a few key steps:

1. Identifying the problem definition clearly

2. Setting clear objectives
3. Establishing success benchmark
4. Understanding end-user engagement/use
5. Understanding the constraints and limitations of deployment
6. Followed by finally doing the data collection.

Keyword Spotting Example

Keyword Spotting (KWS) is an excellent example to illustrate all of the general steps in action. This technology is critical for voice-enabled interfaces on endpoint devices such as smartphones. Typically functioning as lightweight wake-word engines, KWS systems are consistently active, listening for a specific phrase to trigger further actions.

As shown in Figure 6.4, when we say "OK, Google" or "Alexa," this initiates a process on a microcontroller embedded within the device.

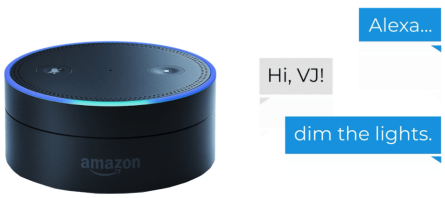


Figure 6.4: Keyword Spotting example: interacting with Alexa. Source: Amazon.

Building a reliable KWS model is a complex task. It demands a deep understanding of the deployment scenario, encompassing where and how these devices will operate. For instance, a KWS model's effectiveness is not just about recognizing a word; it's about discerning it among various accents and background noises, whether in a bustling cafe or amid the blaring sound of a television in a living room or a kitchen where these devices are commonly found. It's about ensuring that a whispered "Alexa" in the dead of night or a shouted "OK Google" in a noisy marketplace are recognized with equal precision.

Moreover, many current KWS voice assistants support a limited number of languages, leaving a substantial portion of the world's linguistic diversity unrepresented. This limitation is partly due to the difficulty in gathering and monetizing data for languages spoken by smaller populations. The long-tail distribution of languages implies that many languages have limited data, making the development of supportive technologies challenging.

This level of accuracy and robustness hinges on the availability and quality of data, the ability to label the data correctly, and the transparency of the data for the end user before it is used to train the model. However, it all begins with clearly understanding the problem statement or definition. Using this KWS as an example, we can break each of the steps out as follows:

1. **Identifying the Problem:** KWS detects specific keywords amidst ambient sounds and other spoken words. The primary problem is to design a system that can recognize these keywords with high accuracy, low latency, and minimal false positives or negatives, especially when deployed on devices with limited computational resources.

2. **Setting Clear Objectives:** The objectives for a KWS system might include:
   - Achieving a specific accuracy rate (e.g., 98% accuracy in keyword detection).
   - Ensuring low latency (e.g., keyword detection and response within 200 milliseconds).
   - Minimizing power consumption to extend battery life on embedded devices.
   - Ensuring the model's size is optimized for the available memory on the device.
3. **Benchmarks for Success:** Establish clear metrics to measure the success of the KWS system. This could include:
   - *True Positive Rate:* The percentage of correctly identified keywords.
   - *False Positive Rate:* The percentage of non-keywords incorrectly identified as keywords.
   - *Response Time:* The time taken from keyword utterance to system response.
   - *Power Consumption:* Average power used during keyword detection.
4. **Stakeholder Engagement and Understanding:** Engage with stakeholders, which include device manufacturers, hardware and software developers, and end-users. Understand their needs, capabilities, and constraints. For instance:
   - Device manufacturers might prioritize low power consumption.
   - Software developers might emphasize ease of integration.
   - End-users would prioritize accuracy and responsiveness.
5. **Understanding the Constraints and Limitations of Embedded Systems:** Embedded devices come with their own set of challenges:
   - *Memory Limitations:* KWS models must be lightweight to fit within the memory constraints of embedded devices. Typically, KWS models need to be as small as 16KB to fit in the always-on island of the SoC. Moreover, this is just the model size. Additional application code for preprocessing may also need to fit within the memory constraints.
   - Processing Power: The computational capabilities of embedded devices are limited (a few hundred MHz of clock speed), so the KWS model must be optimized for efficiency.
   - *Power Consumption:* Since many embedded devices are battery-powered, the KWS system must be power-efficient.
   - *Environmental Challenges:* Devices might be deployed in various environments, from quiet bedrooms to noisy industrial settings. The KWS system must be robust enough to function effectively across these scenarios.

6. **Data Collection and Analysis:** For a KWS system, the quality and diversity of data are paramount. Considerations might include:
    - *Variety of Accents:* Collect data from speakers with various accents to ensure wide-ranging recognition.
    - *Background Noises:* Include data samples with different ambient noises to train the model for real-world scenarios.
    - *Keyword Variations:* People might either pronounce keywords differently or have slight variations in the wake word itself. Ensure the dataset captures these nuances.
7. **Iterative Feedback and Refinement:** Once a prototype KWS system is developed, it is important to do the following to ensure that the system remains aligned with the defined problem and objectives as the deployment scenarios change over time as things evolve.
    - Test it in real-world scenarios
    - Gather feedback
    - Iteratively refine the model

The KWS example illustrates the broader principles of problem definition, showing how initial decisions about data requirements ripple throughout a project's lifecycle. By carefully considering each aspect—from core problem identification through performance benchmarks to deployment constraints—teams can build a strong foundation for their ML systems. The methodical problem definition process provides a framework applicable across the ML spectrum. Whether developing computer vision systems for medical diagnostics, recommendation engines processing millions of user interactions, or natural language models analyzing diverse text corpora, this structured approach helps teams anticipate and plan for their data needs.

This brings us to data pipelines—the foundational infrastructure that transforms raw data into ML—ready formats while maintaining quality and reliability throughout the process. These pipelines implement our carefully defined requirements in production systems, handling everything from initial data ingestion to final feature generation.

Caution 1: Keyword Spotting with TensorFlow Lite Micro

Explore a hands-on guide for building and deploying Keyword Spotting systems using TensorFlow Lite Micro. Follow steps from data collection to model training and deployment to microcontrollers. Learn to create efficient KWS models that recognize specific keywords amidst background noise. Perfect for those interested in machine learning on embedded systems. Unlock the potential of voice-enabled devices with TensorFlow Lite Micro!

6.3 Pipeline Fundamentals

Modern machine learning (ML) systems depend on data pipelines to process massive amounts of data efficiently and reliably. For instance, recommendation systems at companies like Netflix process billions of user interactions daily, while autonomous vehicle systems must handle terabytes of sensor data in real-time. These pipelines serve as the backbone of ML systems, acting as the infrastructure through which raw data transforms into ML-ready training data.

These data pipelines are not simple linear paths but rather complex systems. They must manage data acquisition, transformation, storage, and delivery while ensuring data quality and system reliability. The design of these pipelines fundamentally shapes what is possible with an ML system.

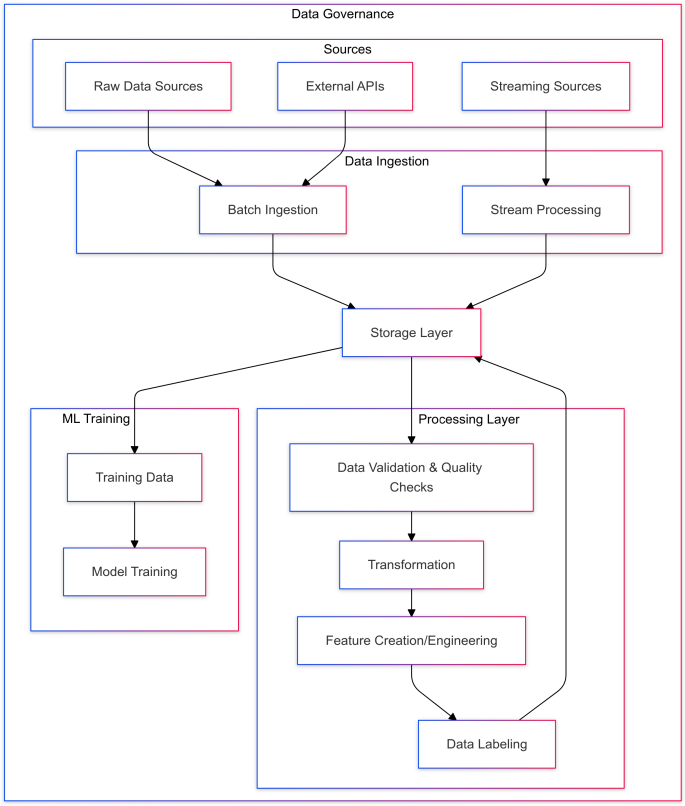


Figure 6.5: Overview of the data pipeline.

ML data pipelines consist of several distinct layers: data sources, ingestion, processing, labeling, storage, and eventually ML training (Figure 6.5). Each layer plays a specific role in the data preparation workflow. The interactions between these layers are crucial to the system's overall effectiveness. The flow from raw data sources to ML training demonstrates the importance of

maintaining data quality and meeting system requirements throughout the pipeline.

6.4 Data Sources

The first stage of the pipeline architecture sourcing appropriate data to meet the training needs. The quality and diversity of our this data will fundamentally determine our ML system's learning and prediction capabilities and limitations. ML systems can obtain their training data through several different approaches, each with their own advantages and challenges. Let's examine each of these approaches in detail.

6.4.1 Pre-existing datasets

Platforms like Kaggle and UCI Machine Learning Repository provide ML practitioners with ready-to-use datasets that can jumpstart system development. These pre-existing datasets are particularly valuable when building ML systems as they offer immediate access to cleaned, formatted data with established benchmarks. One of their primary advantages is cost efficiency – creating datasets from scratch requires significant time and resources, especially when building production ML systems that need large amounts of high-quality training data.

Many of these datasets, such as ImageNet, have become standard benchmarks in the machine learning community, enabling consistent performance comparisons across different models and architectures. For ML system developers, this standardization provides clear metrics for evaluating model improvements and system performance. The immediate availability of these datasets allows teams to begin experimentation and prototyping without delays in data collection and preprocessing.

However, ML practitioners must carefully consider the quality assurance aspects of pre-existing datasets. For instance, the ImageNet dataset was found to have over 6.4% errors (Northcutt, Athalye, and Mueller 2021). While popular datasets benefit from community scrutiny that helps identify and correct errors and biases, these issues can significantly impact system performance if not properly addressed. Moreover, as (Gebru et al. 2021) highlighted in her paper, simply providing a dataset without documentation can lead to misuse and misinterpretation, potentially amplifying biases present in the data.

Supporting documentation often accompanying existing datasets is invaluable, though this generally applies only to widely used datasets. Good documentation provides insights into the data collection process and variable definitions and sometimes even offers baseline model performances. This information not only aids understanding but also promotes reproducibility in research, a cornerstone of scientific integrity; currently, there is a crisis around improving reproducibility in machine learning systems (Pineau et al. 2021). When other researchers have access to the same data, they can validate findings, test new hypotheses, or apply different methodologies, thus allowing us to build on each other's work more rapidly.

While platforms like Kaggle and UCI Machine Learning Repository are invaluable resources, it's essential to understand the context in which the data

was collected. Researchers should be wary of potential overfitting when using popular datasets, as multiple models might have been trained on them, leading to inflated performance metrics. Sometimes, these datasets do not reflect the real-world data.

A key consideration for ML systems is how well pre-existing datasets reflect real-world deployment conditions. Relying on standard datasets can create a concerning disconnect between training and production environments. This misalignment becomes particularly problematic when multiple ML systems are trained on the same datasets (Figure 6.6), potentially propagating biases and limitations throughout an entire ecosystem of deployed models.

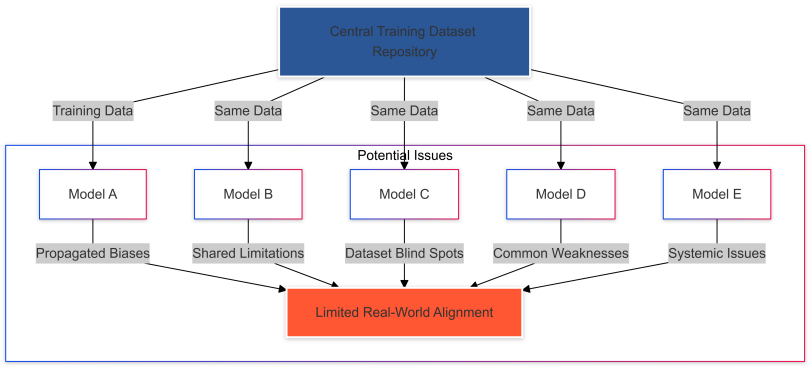


Figure 6.6: Training different models on the same dataset.

6.4.2 Web Scraping

When building ML systems, particularly in domains where pre-existing datasets are insufficient, web scraping offers a powerful approach to gathering training data at scale. This automated technique for extracting data from websites has become a powerful tool in modern ML system development. It enables teams to build custom datasets tailored to their specific needs.

Web scraping has proven particularly valuable for building large-scale ML systems when human-labeled data is scarce. Consider computer vision systems: major datasets like ImageNet and OpenImages were built through systematic web scraping, fundamentally advancing the field of computer vision. In production environments, companies regularly scrape e-commerce sites to gather product images for recognition systems or social media platforms for computer vision applications. Stanford's LabelMe project demonstrated this approach's potential early on, scraping Flickr to create a diverse dataset of over 63,000 annotated images.

The impact of web scraping extends well beyond computer vision systems. In natural language processing, web-scraped data has enabled the development of increasingly sophisticated ML systems. Large language models, such as ChatGPT and Claude, rely on vast amounts of text scraped from the public internet to learn language patterns and generate responses. Similarly, specialized ML systems like GitHub's Copilot demonstrate how

targeted web scraping—in this case, of code repositories—can create powerful domain-specific assistants.

Production ML systems often require continuous data collection to maintain relevance and performance. Web scraping facilitates this by gathering structured data like stock prices, weather patterns, or product information for analytical applications. However, this continuous collection introduces unique challenges for ML systems. Data consistency becomes crucial—variations in website structure or content formatting can disrupt the data pipeline and affect model performance. Proper data management through databases or warehouses becomes essential not just for storage, but for maintaining data quality and enabling model updates.

Despite its utility, web scraping presents several challenges that ML system developers must carefully consider. Legal and ethical constraints can limit data collection—not all websites permit scraping, and violating these restrictions can have serious consequences. When building ML systems with scraped data, teams must carefully document data sources and ensure compliance with terms of service and copyright laws. Privacy considerations become particularly critical when dealing with user-generated content, often requiring robust anonymization procedures.

Technical limitations also affect the reliability of web-scraped training data. Rate limiting by websites can slow data collection, while the dynamic nature of web content can introduce inconsistencies that impact model training. As shown in Figure 6.7, web scraping can yield unexpected or irrelevant data—such as historical images appearing in contemporary image searches—that can pollute training datasets and degrade model performance. These issues highlight the importance of thorough data validation and cleaning processes in ML pipelines built on web-scraped data.

Figure 6.7: A picture of old traffic lights (1914). Source: Vox.

Caution 2: Web Scraping

Discover the power of web scraping with Python using libraries like Beautiful Soup and Pandas. This exercise will scrape Python documentation for function names and descriptions and explore NBA player stats. By the end, you'll have the skills to extract and analyze data from real-world websites. Ready to dive in? Access the Google Colab notebook below and start practicing!

6.4.3 Crowdsourcing

Crowdsourcing is a collaborative approach to data collection, leveraging the collective efforts of distributed individuals via the internet to tackle tasks requiring human judgment. By engaging a global pool of contributors, this method accelerates the creation of high-quality, labeled datasets for machine learning systems, especially in scenarios where pre-existing data is scarce or domain-specific. Platforms like Amazon Mechanical Turk exemplify how crowdsourcing facilitates this process by distributing annotation tasks to a global workforce. This enables the rapid collection of labels for complex tasks such as sentiment analysis, image recognition, and speech transcription, significantly expediting the data preparation phase.

One of the most impactful examples of crowdsourcing in machine learning is the creation of the ImageNet dataset. ImageNet, which revolutionized computer vision, was built by distributing image labeling tasks to contributors via Amazon Mechanical Turk. The contributors categorized millions of images into thousands of classes, enabling researchers to train and benchmark models for a wide variety of visual recognition tasks.

The dataset's availability spurred advancements in deep learning, including the breakthrough AlexNet model in 2012, which demonstrated how large-scale, crowdsourced datasets could drive innovation. ImageNet's success highlights how leveraging a diverse group of contributors for annotation can enable machine learning systems to achieve unprecedented performance.

Another example of crowdsourcing's potential is Google's Crowdsource, a platform where volunteers contribute labeled data to improve AI systems in applications like language translation, handwriting recognition, and image understanding. By gamifying the process and engaging global participants, Google harnesses diverse datasets, particularly for underrepresented languages. This approach not only enhances the quality of AI systems but also empowers communities by enabling their contributions to influence technological development.

Crowdsourcing has also been instrumental in applications beyond traditional dataset annotation. For instance, the navigation app Waze uses crowdsourced data from its users to provide real-time traffic updates, route suggestions, and incident reporting. While this involves dynamic data collec-

tion rather than static dataset labeling, it demonstrates how crowdsourcing can generate continuously updated datasets essential for applications like mobile or edge ML systems. These systems often require real-time input to maintain relevance and accuracy in changing environments.

One of the primary advantages of crowdsourcing is its scalability. By distributing microtasks to a large audience, projects can process enormous volumes of data quickly and cost-effectively. This scalability is particularly beneficial for machine learning systems that require extensive datasets to achieve high performance. Additionally, the diversity of contributors introduces a wide range of perspectives, cultural insights, and linguistic variations, enriching datasets and improving models' ability to generalize across populations.

Flexibility is a key benefit of crowdsourcing. Tasks can be adjusted dynamically based on initial results, allowing for iterative improvements in data collection. For example, Google's reCAPTCHA system uses crowdsourcing to verify human users while simultaneously labeling datasets for training machine learning models. Users identify objects in images—such as street signs or cars—contributing to the training of autonomous systems. This clever integration demonstrates how crowdsourcing can scale seamlessly when embedded into everyday workflows.

Despite its advantages, crowdsourcing presents challenges that require careful management. Quality control is a major concern, as the variability in contributors' expertise and attention can lead to inconsistent or inaccurate annotations. Providing clear instructions and training materials helps ensure participants understand the task requirements. Techniques such as embedding known test cases, leveraging consensus algorithms, or using redundant annotations can mitigate quality issues and align the process with the problem definition discussed earlier.

Ethical considerations are paramount in crowdsourcing, especially when datasets are built at scale using global contributors. It is essential to ensure that participants are fairly compensated for their work and that they are informed about how their contributions will be used. Additionally, privacy concerns must be addressed, particularly when dealing with sensitive or personal information. Transparent sourcing practices, clear communication with contributors, and robust auditing mechanisms are crucial for building trust and maintaining ethical standards.

The issue of fair compensation and ethical data sourcing was brought into sharp focus during the development of large-scale AI systems like OpenAI's ChatGPT. Reports revealed that OpenAI outsourced data annotation tasks to workers in Kenya, employing them to moderate content and identify harmful or inappropriate material that the model might generate. This involved reviewing and labeling distressing content, such as graphic violence and explicit material, to train the AI in recognizing and avoiding such outputs. While this approach enabled OpenAI to improve the safety and utility of ChatGPT, significant ethical concerns arose around the working conditions, the nature of the tasks, and the compensation provided to Kenyan workers.

Many of the contributors were reportedly paid as little as $1.32 per hour for reviewing and labeling highly traumatic material. The emotional toll of such work, coupled with low wages, raised serious questions about the fairness and

transparency of the crowdsourcing process. This controversy highlights a critical gap in ethical crowdsourcing practices. The workers, often from economically disadvantaged regions, were not adequately supported to cope with the psychological impact of their tasks. The lack of mental health resources and insufficient compensation underscored the power imbalances that can emerge when outsourcing data annotation tasks to lower-income regions.

The challenges highlighted by the ChatGPT—Kenya controversy are not unique to OpenAI. Many organizations that rely on crowdsourcing for data annotation face similar issues. As machine learning systems grow more complex and require larger datasets, the demand for annotated data will continue to increase. This shows the need for industry-wide standards and best practices to ensure ethical data sourcing. This case emphasizes the importance of considering the human labor behind AI systems. While crowdsourcing offers scalability and diversity, it also brings ethical responsibilities that cannot be overlooked. Organizations must prioritize the well-being and fair treatment of contributors as they build the datasets that drive AI innovation.

Moreover, when dealing with specialized applications like mobile ML, edge ML, or cloud ML, additional challenges may arise. These applications often require data collected from specific environments or devices, which can be difficult to gather through general crowdsourcing platforms. For example, data for mobile applications utilizing smartphone sensors may necessitate participants with specific hardware features or software versions. Similarly, edge ML systems deployed in industrial settings may require data involving proprietary processes or secure environments, introducing privacy and accessibility challenges.

Hybrid approaches that combine crowdsourcing with other data collection methods can address these challenges. Organizations may engage specialized communities, partner with relevant stakeholders, or create targeted initiatives to collect domain-specific data. Additionally, synthetic data generation, as discussed in the next section, can augment real-world data when crowdsourcing falls short.

6.4.4 Synthetic Data

Synthetic data generation has emerged as a powerful tool for addressing limitations in data collection, particularly in machine learning applications where real-world data is scarce, expensive, or ethically challenging to obtain. This approach involves creating artificial data using algorithms, simulations, or generative models to mimic real-world datasets. The generated data can be used to supplement or replace real-world data, expanding the possibilities for training robust and accurate machine learning systems. Figure 6.8 illustrates the process of combining synthetic data with historical datasets to create larger, more diverse training sets.

Advancements in generative modeling techniques, such as Generative Adversarial Networks (GANs)[1] and Variational Autoencoders (VAEs)[2] , have greatly enhanced the quality of synthetic data. These techniques can produce data that closely resembles real-world distributions, making it suitable for applications ranging from computer vision to natural language processing.

1 **Generative Adversarial Networks (GANs):** Machine learning models with a generator creating data and a discriminator assessing its realism.

2 **Variational Autoencoders (VAEs):** Generative models that encode data into a latent space and decode it to generate new samples.

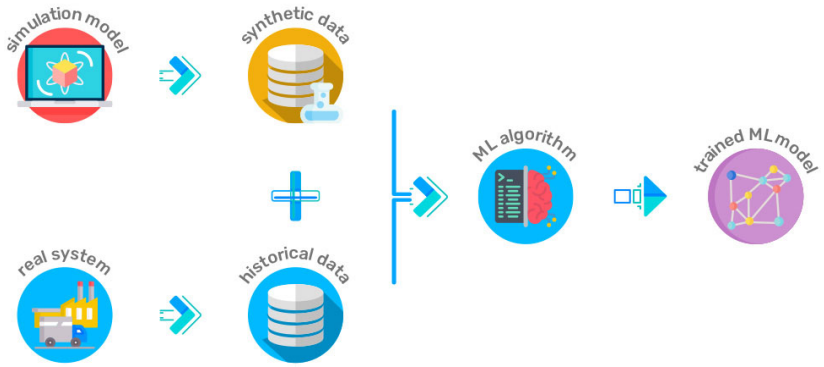


Figure 6.8: Increasing training data size with synthetic data generation. Source: AnyLogic.

For example, GANs have been used to generate synthetic images for object recognition tasks, creating diverse datasets that are almost indistinguishable from real-world images. Similarly, synthetic data has been leveraged to simulate speech patterns, enhancing the robustness of voice recognition systems.

This labeling style makes the footnotes more descriptive and easier to manage in larger documents. Synthetic data has become particularly valuable in domains where obtaining real-world data is either impractical or costly. In security applications, for instance, training a system to detect the sound of breaking glass would require physically breaking numerous windows under controlled conditions. Synthetic data provides a practical alternative by simulating these sounds, allowing the model to learn effectively without the logistical challenges of real-world collection. In healthcare, privacy regulations such as GDPR[3] and HIPAA[4] limit the sharing of sensitive patient information. Synthetic data generation enables the creation of realistic yet anonymized datasets that can be used for training diagnostic models without compromising patient privacy.

The automotive industry has also embraced synthetic data to train autonomous vehicle systems; there are only so many cars you can physically crash to get crash-test data that might help an ML system know how to avoid crashes in the first place. Capturing real-world scenarios, especially rare edge cases such as near-accidents or unusual road conditions, is inherently difficult. Synthetic data allows researchers to simulate these scenarios in a controlled virtual environment, ensuring that models are trained to handle a wide range of conditions. This approach has proven invaluable for advancing the capabilities of self-driving cars.

Another important application of synthetic data lies in augmenting existing datasets. Introducing variations into datasets enhances model robustness by exposing the model to diverse conditions. For instance, in speech recognition, data augmentation techniques like SpecAugment introduce noise, shifts, or pitch variations, enabling models to generalize better across different environments and speaker styles. This principle extends to other domains as well, where synthetic data can fill gaps in underrepresented scenarios or edge cases.

3 **General Data Protection Regulation (GDPR):** A regulation in EU law on data protection and privacy in the European Union and the European Economic Area.

4 **Health Insurance Portability and Accountability Act (HIPAA):** A US law designed to provide privacy standards to protect patients' medical records and other health information.

In addition to expanding datasets, synthetic data addresses critical ethical and privacy concerns. Unlike real-world data, synthetic data is artificially generated and does not tie back to specific individuals or entities. This makes it especially useful in sensitive domains such as finance, healthcare, or human resources, where data confidentiality is paramount. The ability to preserve statistical properties while removing identifying information allows researchers to maintain high ethical standards without compromising the quality of their models.

Poorly generated data can misrepresent underlying real-world distributions, introducing biases or inaccuracies that degrade model performance. Validating synthetic data against real-world benchmarks is essential to ensure its reliability. Additionally, models trained primarily on synthetic data must be rigorously tested in real-world scenarios to confirm their ability to generalize effectively. Another challenge is the potential amplification of biases present in the original datasets used to inform synthetic data generation. If these biases are not carefully addressed, they may be inadvertently reinforced in the resulting models.

Synthetic data has revolutionized the way machine learning systems are trained, providing flexibility, diversity, and scalability in data preparation. However, as its adoption grows, practitioners must remain vigilant about its limitations and ethical implications. By combining synthetic data with rigorous validation and thoughtful application, machine learning researchers and engineers can unlock its full potential while ensuring reliability and fairness in their systems.

Caution 3: Synthetic Data

Let us learn about synthetic data generation using GANs on tabular data. We'll take a hands-on approach, diving into the workings of the CTGAN model and applying it to the Synthea dataset from the healthcare domain. From data preprocessing to model training and evaluation, we'll go step-by-step, learning how to create synthetic data, assess its quality, and unlock the potential of GANs for data augmentation and real-world applications.

6.4.5 Case Study: KWS

KWS is an excellent case study of how different data collection approaches can be combined effectively. Each method we've discussed plays a role in building robust wake word detection systems, albeit with different trade-offs:

Pre-existing datasets like Google's Speech Commands (Warden 2018) provide a foundation for initial development, offering carefully curated voice samples for common wake words. However, these datasets often lack diversity in accents, environments, and languages, necessitating additional data collection strategies.

Web scraping can supplement these baseline datasets by gathering diverse voice samples from video platforms, podcast repositories, and speech databases. This helps capture natural speech patterns and wake word variations, though careful attention must be paid to audio quality and privacy considerations when scraping voice data.

Crowdsourcing becomes valuable for collecting specific wake word samples across different demographics and environments. Platforms like Amazon Mechanical Turk can engage contributors to record wake words in various accents, speaking styles, and background conditions. This approach is particularly useful for gathering data for underrepresented languages or specific acoustic environments.

Synthetic data generation helps fill remaining gaps by creating unlimited variations of wake word utterances. Using speech synthesis and audio augmentation techniques, developers can generate training data that captures different acoustic environments (busy streets, quiet rooms, moving vehicles), speaker characteristics (age, accent, gender), and background noise conditions.

This multi-faceted approach to data collection enables the development of KWS systems that perform robustly across diverse real-world conditions. The combination of methods helps address the unique challenges of wake word detection, from handling various accents and background noise to maintaining consistent performance across different devices and environments.

6.5 Data Ingestion

The collected data must be reliably and efficiently ingested into our ML systems through well-designed data pipelines. This transformation presents several challenges that ML engineers must address.

6.5.1 Ingestion Patterns

In ML systems, data ingestion typically follows two primary patterns: batch ingestion and stream ingestion. Each pattern has distinct characteristics and use cases that students should understand to design effective ML systems.

Batch ingestion involves collecting data in groups or batches over a specified period before processing. This method is appropriate when real-time data processing is not critical and data can be processed at scheduled intervals. It's also useful for loading large volumes of historical data. For example, a retail company might use batch ingestion to process daily sales data overnight, updating their ML models for inventory prediction each morning (Akidau et al. 2015).

In contrast, stream ingestion processes data in real-time as it arrives. This pattern is crucial for applications requiring immediate data processing, scenarios where data loses value quickly, and systems that need to respond to events as they occur. A financial institution, for instance, might use stream ingestion for real-time fraud detection, processing each transaction as it occurs to flag suspicious activity immediately (Kleppmann 2016).

Many modern ML systems employ a hybrid approach, combining both batch and stream ingestion to handle different data velocities and use cases. This

flexibility allows systems to process both historical data in batches and real-time data streams, providing a comprehensive view of the data landscape.

6.5.2 ETL vs. ELT

When designing data ingestion pipelines for ML systems, it's necessary to understand the differences between Extract, Transform, Load (ETL) and Extract, Load, Transform (ELT) approaches. These paradigms determine when data transformations occur relative to the loading phase, significantly impacting the flexibility and efficiency of your ML pipeline.

ETL is a well-established paradigm in which data is first gathered from a source, then transformed to match the target schema or model, and finally loaded into a data warehouse or other repository. This approach typically results in data being stored in a ready-to-query format, which can be advantageous for ML systems that require consistent, pre-processed data. For instance, an ML system predicting customer churn might use ETL to standardize and aggregate customer interaction data from multiple sources before loading it into a format suitable for model training (Inmon 2005).

However, ETL can be less flexible when schemas or requirements change frequently, a common occurrence in evolving ML projects. This is where the ELT approach comes into play. ELT reverses the order by first loading raw data and then applying transformations as needed. This method is often seen in modern data lake or schema-on-read[5] environments, allowing for a more agile approach when addressing evolving analytical needs in ML systems.

[5] **Schema-on-read:** A flexible approach where data structure is defined at access time, not during ingestion, enabling versatile use of raw data.

By deferring transformations, ELT can accommodate varying uses of the same dataset, which is particularly useful in exploratory data analysis phases of ML projects or when multiple models with different data requirements are being developed simultaneously. However, it's important to note that ELT places greater demands on storage systems and query engines, which must handle large amounts of unprocessed information.

In practice, many ML systems employ a hybrid approach, selecting ETL or ELT on a case-by-case basis depending on the specific requirements of each data source or ML model. For example, a system might use ETL for structured data from relational databases where schemas are well-defined and stable, while employing ELT for unstructured data like text or images where transformation requirements may evolve as the ML models are refined.

6.5.3 Source Integration

Integrating diverse data sources is a key challenge in data ingestion for ML systems. Data may come from various origins, including databases, APIs, file systems, and IoT devices. Each source may have its own data format, access protocol, and update frequency.

To effectively integrate these sources, ML engineers must develop robust connectors or adapters for each data source. These connectors handle the specifics of data extraction, including authentication, rate limiting, and error handling. For example, when integrating with a REST API, the connector would manage API keys, respect rate limits, and handle HTTP status codes appropriately.

Furthermore, source integration often involves data transformation at the ingestion point. This might include parsing JSON or XML responses, converting timestamps to a standard format, or performing basic data cleaning operations. The goal is to standardize the data format as it enters the ML pipeline, simplifying downstream processing.

It's also essential to consider the reliability and availability of data sources. Some sources may experience downtime or have inconsistent data quality. Implementing retry mechanisms, data quality checks, and fallback procedures can help ensure a steady flow of reliable data into the ML system.

6.5.4 Data Validation

Data validation is an important step in the ingestion process, ensuring that incoming data meets quality standards and conforms to expected schemas. This step helps prevent downstream issues in ML pipelines caused by data anomalies or inconsistencies.

At the ingestion stage, validation typically encompasses several key aspects. First, it checks for schema conformity, ensuring that incoming data adheres to the expected structure, including data types and field names. Next, it verifies data ranges and constraints, confirming that numeric fields fall within expected ranges and that categorical fields contain valid values. Completeness checks are also performed, looking for missing or null values in required fields. Additionally, consistency checks ensure that related data points are logically coherent (Gudivada, Rao, et al. 2017).

For example, in a healthcare ML system ingesting patient data, validation might include checking that age values are positive integers, diagnosis codes are from a predefined set, and admission dates are not in the future. By implementing robust validation at the ingestion stage, ML engineers can detect and handle data quality issues early, significantly reducing the risk of training models on flawed or inconsistent data.

6.5.5 Error Handling

Error handling in data ingestion is essential for building resilient ML systems. Errors can occur at various points in the ingestion process, from source connection issues to data validation failures. Effective error handling strategies ensure that the ML pipeline can continue to operate even when faced with data ingestion challenges.

A key concept in error handling is graceful degradation. This involves designing systems to continue functioning, possibly with reduced capabilities, when faced with partial data loss or temporary source unavailability. Implementing intelligent retry logic for transient errors, such as network interruptions or temporary service outages, is another important aspect of robust error handling. Many ML systems employ the concept of dead letter queues[6] , using separate storage for data that fails processing. This allows for later analysis and potential reprocessing of problematic data (Kleppmann 2016).

For instance, in a financial ML system ingesting market data, error handling might involve falling back to slightly delayed data sources if real-time feeds fail, while simultaneously alerting the operations team to the issue. This approach

[6] **Dead Letter Queues:** Queues that store unprocessed messages for analysis or reprocessing.

ensures that the system continues to function and that responsible parties are aware of and can address the problem.

This ensures that downstream processes have access to reliable, high-quality data for training and inference tasks, even in the face of ingestion challenges. Understanding these concepts of data validation and error handling is essential for students and practitioners aiming to build robust, production-ready ML systems.

Once ingestion is complete and data is validated, it is typically loaded into a storage environment suited to the organization's analytical or machine learning needs. Some datasets flow into data warehouses for structured queries, whereas others are retained in data lakes for exploratory or large-scale analyses. Advanced systems may also employ feature stores to provide standardized features for machine learning.

6.5.6 Case Study: KWS

A production KWS system typically employs both streaming and batch ingestion patterns. The streaming pattern handles real-time audio data from active devices, where wake words must be detected with minimal latency. This requires careful implementation of pub/sub mechanisms—for example, using Apache Kafka-like streams to buffer incoming audio data and enable parallel processing across multiple inference servers.

Simultaneously, the system processes batch data for model training and updates. This includes ingesting new wake word recordings from crowdsourcing efforts, synthetic data from voice generation systems, and validated user interactions. The batch processing typically follows an ETL pattern, where audio data is preprocessed (normalized, filtered, segmented) before being stored in a format optimized for model training.

KWS systems must integrate data from diverse sources, such as real-time audio streams from deployed devices, crowdsourced recordings from data collection platforms etc. Each source presents unique challenges. Real-time audio streams require rate limiting to prevent system overload during usage spikes. Crowdsourced data needs robust validation to ensure recording quality and correct labeling. Synthetic data must be verified for realistic representation of wake word variations.

KWS systems employ sophisticated error handling mechanisms due to the nature of voice interaction. When processing real-time audio, dead letter queues store failed recognition attempts for analysis, helping identify patterns in false negatives or system failures. Data validation becomes particularly important for maintaining wake word detection accuracy—incoming audio must be checked for quality issues like clipping, noise levels, and appropriate sampling rates.

For example, consider a smart home device processing the wake word "Alexa." The ingestion pipeline must validate:

- Audio quality metrics (signal-to-noise ratio, sample rate, bit depth)
- Recording duration (typically 1-2 seconds for wake words)
- Background noise levels

- Speaker proximity indicators

Invalid samples are routed to dead letter queues for analysis, while valid samples are processed in real-time for wake word detection.

This case study illustrates how real-world ML systems must carefully balance different ingestion patterns, handle multiple data sources, and maintain robust error handling—all while meeting strict latency and reliability requirements. The lessons from KWS systems apply broadly to other ML applications requiring real-time processing capabilities alongside continuous model improvement.

6.6 Data Processing

Data processing is a stage in the machine learning pipeline that transforms raw data into a format suitable for model training and inference. This stage encompasses several key activities, each playing a role in preparing data for effective use in ML systems. The approach to data processing is closely tied to the ETL (Extract, Transform, Load) or ELT (Extract, Load, Transform) paradigms discussed earlier.

In traditional ETL workflows, much of the data processing occurs before the data is loaded into the target system. This approach front-loads the cleaning, transformation, and feature engineering steps, ensuring that data is in a ready-to-use state when it reaches the data warehouse or ML pipeline. ETL is often preferred when dealing with structured data or when there's a need for significant data cleansing before analysis.

Conversely, in ELT workflows, raw data is first loaded into the target system, and transformations are applied afterwards. This approach, often used with data lakes, allows for more flexibility in data processing. It's particularly useful when dealing with unstructured or semi-structured data, or when the exact transformations needed are not known in advance. In ELT, many of the data processing steps we'll discuss might be performed on-demand or as part of the ML pipeline itself.

The choice between ETL and ELT can impact how and when data processing occurs in an ML system. For instance, in an ETL-based system, data cleaning and initial transformations might happen before the data even reaches the ML team. In contrast, an ELT-based system might require ML engineers to handle more of the data processing tasks as part of their workflow.

Regardless of whether an organization follows an ETL or ELT approach, understanding the following data processing steps is crucial for ML practitioners. These processes ensure that data is clean, relevant, and optimally formatted for machine learning algorithms.

6.6.1 Data Cleaning

Data cleaning involves identifying and correcting errors, inconsistencies, and inaccuracies in datasets. Raw data frequently contains issues such as missing values, duplicates, or outliers that can significantly impact model performance if left unaddressed.

In practice, data cleaning might involve removing duplicate records, handling missing values through imputation or deletion, and correcting formatting inconsistencies. For instance, in a customer database, names might be inconsistently capitalized or formatted. A data cleaning process would standardize these entries, ensuring that "John Doe," "john doe," and "DOE, John" are all treated as the same entity.

Outlier detection and treatment is another important aspect of data cleaning. Outliers can sometimes represent valuable information about rare events, but they can also be the result of measurement errors or data corruption. ML practitioners must carefully consider the nature of their data and the requirements of their models when deciding how to handle outliers.

6.6.2 Quality Assessment

Quality assessment goes hand in hand with data cleaning, providing a systematic approach to evaluating the reliability and usefulness of data. This process involves examining various aspects of data quality, including accuracy, completeness, consistency, and timeliness.

Tools and techniques for quality assessment range from simple statistical measures to more complex machine learning-based approaches. For example, data profiling tools can provide summary statistics and visualizations that help identify potential quality issues. More advanced techniques might involve using unsupervised learning algorithms to detect anomalies or inconsistencies in large datasets.

Establishing clear quality metrics and thresholds is essential for maintaining data quality over time. These metrics might include the percentage of missing values, the frequency of outliers, or measures of data freshness. Regular quality assessments help ensure that data entering the ML pipeline meets the necessary standards for reliable model training and inference.

6.6.3 Data Transformation

Data transformation converts the data from its raw form into a format more suitable for analysis and modeling. This process can include a wide range of operations, from simple conversions to complex mathematical transformations.

Common transformation tasks include normalization and standardization, which scale numerical features to a common range or distribution. For example, in a housing price prediction model, features like square footage and number of rooms might be on vastly different scales. Normalizing these features ensures that they contribute more equally to the model's predictions (Bishop 2006).

Other transformations might involve encoding categorical variables, handling date and time data, or creating derived features. For instance, one-hot encoding[7] is often used to convert categorical variables into a format that can be readily understood by many machine learning algorithms.

[7] **One-Hot Encoding:** Converts categorical variables into binary vectors, where each category is represented by a unique vector with one element set to 1 and the rest to 0. This allows categorical data to be used in ML models requiring numerical input.

6.6.4 Feature Engineering

Feature engineering is the process of using domain knowledge to create new features that make machine learning algorithms work more effectively. This step is often considered more of an art than a science, requiring creativity and deep understanding of both the data and the problem at hand.

Feature engineering might involve combining existing features, extracting information from complex data types, or creating entirely new features based on domain insights. For example, in a retail recommendation system, engineers might create features that capture the recency, frequency, and monetary value of customer purchases, known as RFM analysis (Kuhn and Johnson 2013).

The importance of feature engineering cannot be overstated. Well-engineered features can often lead to significant improvements in model performance, sometimes outweighing the impact of algorithm selection or hyperparameter tuning.

6.6.5 Processing Pipelines

Processing pipelines bring together the various data processing steps into a coherent, reproducible workflow. These pipelines ensure that data is consistently prepared across training and inference stages, reducing the risk of data leakage and improving the reliability of ML systems.

Modern ML frameworks and tools often provide capabilities for building and managing data processing pipelines. For instance, Apache Beam and TensorFlow Transform allow developers to define data processing steps that can be applied consistently during both model training and serving.

Effective pipeline design involves considerations such as modularity, scalability, and version control. Modular pipelines allow for easy updates and maintenance of individual processing steps. Version control for pipelines is crucial, ensuring that changes in data processing can be tracked and correlated with changes in model performance.

6.6.6 Scale Considerations

As datasets grow larger and ML systems become more complex, the scalability of data processing becomes increasingly important. Processing large volumes of data efficiently often requires distributed computing approaches and careful consideration of computational resources.

Techniques for scaling data processing include parallel processing, where data is divided across multiple machines or processors for simultaneous processing. Distributed frameworks like Apache Spark are commonly used for this purpose, allowing data processing tasks to be scaled across large clusters of computers.

Another important consideration is the balance between preprocessing and on-the-fly computation. While extensive preprocessing can speed up model training and inference, it can also lead to increased storage requirements and potential data staleness. Some ML systems opt for a hybrid approach, preprocessing certain features while computing others on-the-fly during model training or inference.

Effective data processing is fundamental to the success of ML systems. By carefully cleaning, transforming, and engineering data, practitioners can significantly improve the performance and reliability of their models. As the field of machine learning continues to evolve, so too do the techniques and tools for data processing, making this an exciting and dynamic area of study and practice.

6.6.7 Case Study: KWS

A KWS system requires careful cleaning of audio recordings to ensure reliable wake word detection. Raw audio data often contains various imperfections—background noise, clipped signals, varying volumes, and inconsistent sampling rates. For example, when processing the wake word "Alexa," the system must clean recordings to standardize volume levels, remove ambient noise, and ensure consistent audio quality across different recording environments, all while preserving the essential characteristics that make the wake word recognizable.

Building on clean data, quality assessment becomes important for KWS systems. Quality metrics for KWS data are uniquely focused on audio characteristics, including signal-to-noise ratio (SNR), audio clarity scores, and speaking rate consistency. For instance, a KWS quality assessment pipeline might automatically flag recordings where background noise exceeds acceptable thresholds or where the wake word is spoken too quickly or unclearly, ensuring only high-quality samples are used for model training.

These quality metrics must be carefully calibrated to reflect real-world operating conditions. A robust training dataset incorporates both pristine recordings and samples containing controlled levels of environmental variations. For instance, while recordings with signal-masking interference are excluded, the dataset should include samples with measured background acoustics, variable speaker distances, and concurrent speech or other forms of audio signals. This approach to data diversity ensures the model maintains wake word detection reliability across the full spectrum of deployment environments and acoustic conditions.

Once quality is assured, transforming audio data for KWS involves converting raw waveforms into formats suitable for ML models. The typical transformation pipeline converts audio signals into spectrograms[8] or mel-frequency cepstral coefficients (MFCCs)[9] , standardizing the representation across different recording conditions. This transformation must be consistently applied across both training and inference, often with additional considerations for real-time processing on edge devices.

Figure 6.9 illustrates this transformation process. The top panel is a raw waveform of a simulated audio signal, which consists of a sine wave mixed with noise. This time-domain representation highlights the challenges posed by real-world recordings, where noise and variability must be addressed. The middle panel shows the spectrogram of the signal, which maps its frequency content over time. The spectrogram provides a detailed view of how energy is distributed across frequencies, making it easier to analyze patterns that could influence wake word recognition, such as the presence of background noise

[8] **Spectrogram:** A visual representation of the spectrum of frequencies in a signal as it varies over time, commonly used in audio processing.

[9] **Mel-Frequency Cepstral Coefficients (MFCCs):** Features extracted from audio signals that represent the short-term power spectrum, widely used in speech and audio analysis.

or signal distortions The bottom panel shows the MFCCs, derived from the spectrogram. These coefficients compress the audio information into a format that emphasizes speech-related characteristics, making them well-suited for KWS tasks.

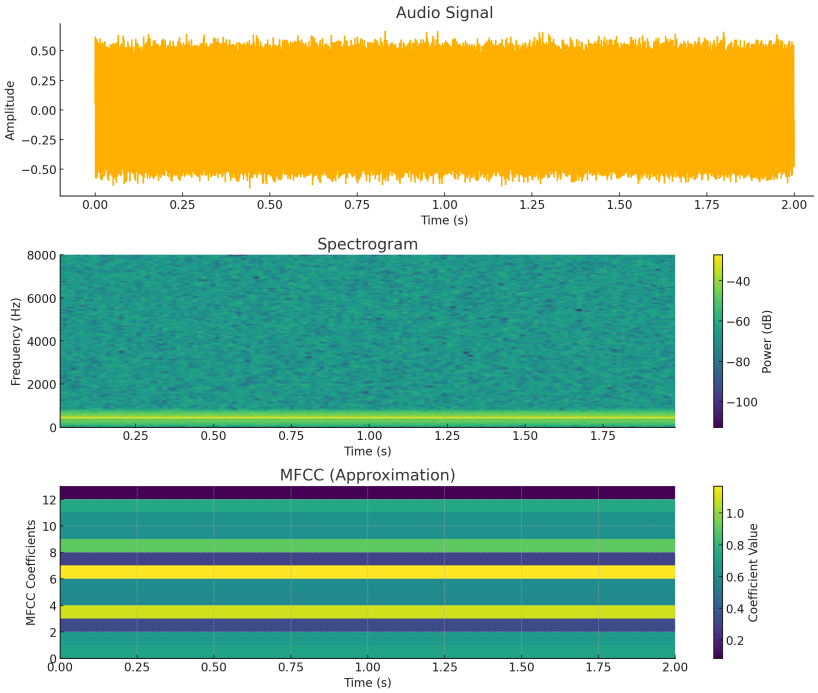


Figure 6.9: KWS data processing of an audio signal (top panel) represented in a spectrogram (middle panel) showing the energy distribution across time and frequency, along with the corresponding MFCCs (bottom panel) that capture perceptually relevant features.

With transformed data in hand, feature engineering for KWS focuses on extracting characteristics that help distinguish wake words from background speech. Engineers might create features capturing tonal variations, speech energy patterns, or temporal characteristics. For the wake word "Alexa," features might include energy distribution across frequency bands, pitch contours, and duration patterns that characterize typical pronunciations.

In practice, bringing all these elements together, KWS processing pipelines must handle both batch processing for training and real-time processing for inference. The pipeline typically includes stages for audio preprocessing, feature extraction, and quality filtering. Importantly, these pipelines must be designed to operate efficiently on edge devices while maintaining consistent processing steps between training and deployment.

6.7 Data Labeling

While data engineering encompasses many aspects of preparing data for machine learning systems, data labeling represents a particularly complex systems challenge. As training datasets grow to millions or billions of examples,

the infrastructure supporting labeling operations becomes increasingly critical to system performance.

Modern machine learning systems must efficiently handle the creation, storage, and management of labels across their data pipeline. The systems architecture must support various labeling workflows while maintaining data consistency, ensuring quality, and managing computational resources effectively. These requirements compound when dealing with large-scale datasets or real-time labeling needs.

The systematic challenges extend beyond just storing and managing labels. Production ML systems need robust pipelines that integrate labeling workflows with data ingestion, preprocessing, and training components. These pipelines must maintain high throughput while ensuring label quality and adapting to changing requirements. For instance, a speech recognition system might need to continuously update its training data with new audio samples and corresponding transcription labels, requiring careful coordination between data collection, labeling, and training subsystems.

Infrastructure requirements vary significantly based on labeling approach and scale. Manual expert labeling may require specialized interfaces and security controls, while automated labeling systems need substantial compute resources for inference. Organizations must carefully balance these requirements against performance needs and resource constraints.

We explore how data labeling fundamentally shapes machine learning system design. From storage architectures to quality control pipelines, each aspect of the labeling process introduces unique technical challenges that ripple throughout the ML infrastructure. Understanding these systems-level implications is essential for building robust, scalable labeling solutions which are an integral part of data negineering.

6.7.1 Label Types

To build effective machine learning systems, we must first understand how different types of labels affect our system architecture and resource requirements. Let's explore this through a practical example: imagine building a smart city system that needs to detect and track various objects like vehicles, pedestrians, and traffic signs from video feeds. Labels capture information about key tasks or concepts.

- **Classification labels** are the simplest form, categorizing images with specific tags (e.g., labeling an image as "car" or "pedestrian"). While conceptually straightforward, a production system processing millions of video frames must efficiently store and retrieve these labels.
- **Bounding boxes** go further by identifying object locations, drawing a box around each object of interest. Our system now needs to track not just what objects exist, but where they are in each frame. This spatial information introduces new storage and processing challenges, especially when tracking moving objects across video frames.
- **Segmentation maps** provide the most detailed information by classifying objects at the pixel level, highlighting each object in a distinct color. For our traffic monitoring system, this might mean precisely outlining

each vehicle, pedestrian, and road sign. These detailed annotations significantly increase our storage and processing requirements.

Figure 6.10 illustrates the common label types:

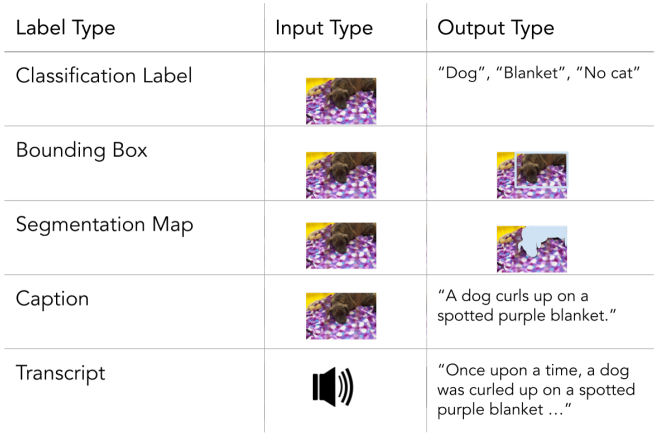


Figure 6.10: An overview of common label types.

The choice of label format depends heavily on our system requirements and resource constraints (Johnson-Roberson et al. 2017). While classification labels might suffice for simple traffic counting, autonomous vehicles need detailed segmentation maps to make precise navigation decisions. Leading autonomous vehicle companies often maintain hybrid systems that store multiple label types for the same data, allowing flexible use across different applications.

Beyond the core labels, production systems must also handle rich metadata. The Common Voice dataset (Ardila et al. 2020), for instance, exemplifies this in its management of audio data for speech recognition. The system tracks speaker demographics for model fairness, recording quality metrics for data filtering, validation status for label reliability, and language information for multilingual support.

Modern labeling platforms have built sophisticated metadata management systems to handle these complex relationships. This metadata becomes important for maintaining and managing data quality and debugging model behavior. If our traffic monitoring system performs poorly in rainy conditions, having metadata about weather conditions during data collection helps identify and address the issue. The infrastructure must efficiently index and query this metadata alongside the primary labels.

The choice of label type cascades through our entire system design. A system built for simple classification labels would need significant modifications to handle segmentation maps efficiently. The infrastructure must optimize storage systems for the chosen label format, implement efficient data retrieval patterns for training, maintain quality control pipelines for validation, and manage version control for label updates. Resource allocation becomes particularly critical as data volume grows, requiring careful capacity planning across storage, compute, and networking components.

6.7.2 Annotation Methods

Manual labeling by experts is the primary approach in many annotation pipelines. This method produces high-quality results but also raises considerable system design challenges. For instance, in medical imaging systems, experienced radiologists offer essential annotations. Such systems necessitate specialized interfaces for accurate labeling, secure data access controls to protect patient privacy, and reliable version control mechanisms to monitor annotation revisions. Despite the dependable outcomes of expert labeling, the scarcity and high expenses of specialists render it challenging to implement on a large scale for extensive datasets.

As we discussed earlier, crowdsourcing offers a path to greater scalability by distributing annotation tasks across many annotators (Sheng and Zhang 2019). Crowdsourcing enables non-experts to distribute annotation tasks, often through dedicated platforms (Sheng and Zhang 2019). Several companies have emerged as leaders in this space, building sophisticated platforms for large-scale annotation. For instance, companies such as Scale AI specialize in managing thousands of concurrent annotators through their platform. Appen focuses on linguistic annotation and text data, while Labelbox has developed specialized tools for computer vision tasks. These platforms allow dataset creators to access a large pool of annotators, making it possible to label vast amounts of data relatively quickly.

Weakly supervised and programmatic methods represent a third approach, using automation to reduce manual effort (Ratner et al. 2018). These systems leverage existing knowledge bases and heuristics to automatically generate labels. For example, distant supervision techniques might use a knowledge base to label mentions of companies in text data. While these methods can rapidly label large datasets, they require substantial compute resources for inference, sophisticated caching systems to avoid redundant computation, and careful monitoring to manage potential noise and bias.

Most production systems combine multiple annotation approaches to balance speed, cost, and quality. A common pattern employs programmatic labeling for initial coverage, followed by crowdsourced verification and expert review of uncertain cases. This hybrid approach requires careful system design to manage the flow of data between different annotation stages. The infrastructure must track label provenance, manage quality control at each stage, and ensure consistent data access patterns across different annotator types.

The choice of annotation method significantly impacts system architecture. Expert-only systems might employ centralized architectures with high-speed access to a single data store. Crowdsourcing demands distributed architectures to handle concurrent annotators. Automated systems need substantial compute resources and caching infrastructure. Many organizations implement tiered architectures where different annotation methods operate on different subsets of data based on complexity and criticality.

Clear guidelines and thorough training remain essential regardless of the chosen architecture. The system must provide consistent interfaces, documentation, and quality metrics across all annotation methods. This becomes particularly challenging when managing diverse annotator pools with varying levels

of expertise. Some platforms address this by offering access to specialized annotators. For instance, providing medical professionals for healthcare datasets or domain experts for technical content.

6.7.3 Label Quality

Label quality is extremely important for machine learning system performance. A model can only be as good as its training data. However, ensuring quality at scale presents significant systems challenges. The fundamental challenge stems from label uncertainty.

Figure 6.11 illustrates common failure modes in labeling systems: some errors arise from data quality issues (like the blurred frog image), while others require deep domain expertise (as with the black stork identification). Even with clear instructions and careful system design, some fraction of labels will inevitably be incorrect (Northcutt, Athalye, and Mueller 2021).

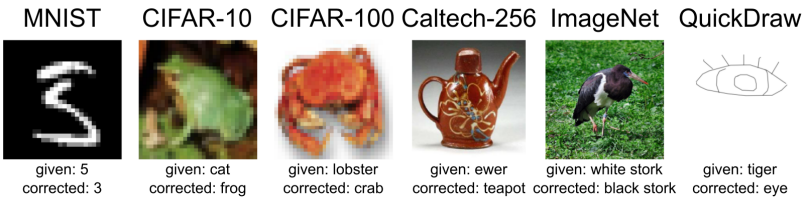


Figure 6.11: Some examples of hard labeling cases. Source: Northcutt, Athalye, and Mueller (2021)

Production ML systems implement multiple layers of quality control to address these challenges. Typically, systematic quality checks continuously monitor the labeling pipeline. These systems randomly sample labeled data for expert review and employ statistical methods to flag potential errors. The infrastructure must efficiently process these checks across millions of examples without creating bottlenecks in the labeling pipeline.

Collecting multiple labels per data point, often referred to as "consensus labeling," can help identify controversial or ambiguous cases. Professional labeling companies have developed sophisticated infrastructure for this process. For example, Labelbox has consensus tools that track inter-annotator agreement rates and automatically route controversial cases for expert review. Scale AI implements tiered quality control, where experienced annotators verify the work of newer team members.

Beyond technical infrastructure, successful labeling systems must consider human factors. When working with annotators, organizations need robust systems for training and guidance. This includes good documentation, clear examples of correct labeling, and regular feedback mechanisms. For complex or domain-specific tasks, the system might implement tiered access levels, routing challenging cases to annotators with appropriate expertise.

Ethical considerations also significantly impact system design. For datasets containing potentially disturbing content, systems should implement protective features like grayscale viewing options (Blackwood et al. 2019). This requires additional image processing pipelines and careful interface design. We need to develop workload management systems that track annotator exposure to sensitive content and enforce appropriate limits.

The quality control system itself generates substantial data that must be efficiently processed and monitored. Organizations typically track inter-annotator agreement rates, label confidence scores, time spent per annotation, error patterns and types, annotator performance metrics, and bias indicators. These metrics must be computed and updated efficiently across millions of examples, often requiring dedicated analytics pipelines.

Regular bias audits are another critical component of quality control. Systems must monitor for cultural, personal, or professional biases that could skew the labeled dataset. This requires infrastructure for collecting and analyzing demographic information, measuring label distributions across different annotator groups, identifying systematic biases in the labeling process, and implementing corrective measures when biases are detected.

Perhaps the most important aspect is that the process must remain iterative. As new challenges emerge, quality control systems must adapt and evolve. Through careful system design and implementation of these quality control mechanisms, organizations can maintain high label quality even at a massive scale.

6.7.4 AI-Assisted Annotation

As machine learning systems grow in scale and complexity, organizations increasingly leverage AI to accelerate and enhance their labeling pipelines. This approach introduces new system design considerations around model deployment, resource management, and human-AI collaboration. The fundamental challenge stems from data volume. Manual annotation alone cannot keep pace with modern ML systems' data needs. As illustrated in Figure 6.12, AI assistance offers several paths to scale labeling operations, each requiring careful system design to balance speed, quality, and resource usage.

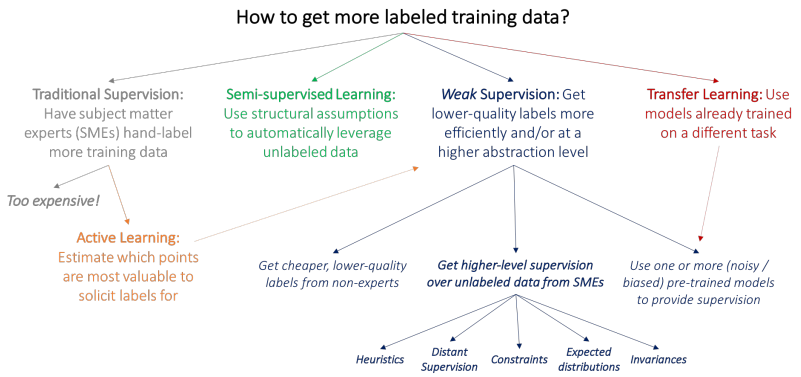


Figure 6.12: Strategies for acquiring additional labeled training data. Source: Stanford AI Lab.

Modern AI-assisted labeling typically employs a combination of approaches. Pre-annotation involves using AI models to generate preliminary labels for a dataset, which humans can then review and correct. Major labeling platforms have made significant investments in this technology. Snorkel AI uses programmatic labeling to automatically generate initial labels at scale. Scale

AI deploys pre-trained models to accelerate annotation in specific domains like autonomous driving, while manycompanies like SuperAnnotate provide automated pre-labeling tools that can reduce manual effort drastically. This method, which often employs semi-supervised learning techniques (Chapelle, Scholkopf, and Zien 2009), can save a significant amount of time, especially for extremely large datasets.

The emergence of Large Language Models (LLMs) like ChatGPT has further transformed labeling pipelines. Beyond simple classification, LLMs can generate rich text descriptions, create labeling guidelines, and even explain their reasoning. For instance, content moderation systems use LLMs to perform initial content classification and generate explanations for policy violations. However, integrating LLMs introduces new system challenges around inference costs, rate limiting, and output validation. Many organizations adopt a tiered approach, using smaller specialized models for routine cases while reserving larger LLMs for complex scenarios.

Methods such as active learning[10] complement these approaches by intelligently prioritizing which examples need human attention. These systems continuously analyze model uncertainty to identify valuable labeling candidates for humans to label. The infrastructure must efficiently compute uncertainty metrics, maintain task queues, and adapt prioritization strategies based on incoming labels. Consider a medical imaging system: active learning might identify unusual pathologies for expert review while handling routine cases automatically.

[10] A machine learning approach where the model selects the most informative data points for labeling to improve learning efficiency.

Quality control becomes increasingly crucial as these AI components interact. The system must monitor both AI and human performance, detect potential errors, and maintain clear label provenance. This requires dedicated infrastructure tracking metrics like model confidence and human-AI agreement rates. In safety-critical domains like self-driving cars, these systems must maintain particularly rigorous standards while processing massive streams of sensor data.

Real-world deployments demonstrate these principles at scale. Medical imaging systems (Krishnan, Rajpurkar, and Topol 2022) combine pre-annotation for common conditions with active learning for unusual cases, all while maintaining strict patient privacy.

Self-driving vehicle systems coordinate multiple AI models to label diverse sensor data in real-time. Social media platforms process millions of items hourly, using tiered approaches where simpler models handle clear cases while complex content routes to more sophisticated models or human reviewers.

While AI assistance offers clear benefits, it also introduces new failure modes. Systems must guard against bias amplification, where AI models trained on biased data perpetuate those biases in new labels. The infrastructure needs robust monitoring to detect such issues and mechanisms to break problematic feedback loops. Human oversight remains essential, requiring careful interface design to help annotators effectively supervise and correct AI output.

6.7.5 Challenges and Limitations

While data labeling is essential for the development of supervised machine learning models, it comes with its own set of challenges and limitations that practitioners must be aware of and address. One of the primary challenges in data labeling is the inherent subjectivity in many labeling tasks. Even with clear guidelines, human annotators may interpret data differently, leading to inconsistencies in labeling. This is particularly evident in tasks involving sentiment analysis, image classification of ambiguous objects, or labeling of complex medical conditions. For instance, in a study of medical image annotation, Oakden-Rayner et al. (2020) found significant variability in labels assigned by different radiologists, highlighting the challenge of obtaining "ground truth" in inherently subjective tasks.

Scalability presents another significant challenge, especially as datasets grow larger and more complex. Manual labeling is time-consuming and expensive, often becoming a bottleneck in the machine learning pipeline. While crowdsourcing and AI-assisted methods can help address this issue to some extent, they introduce their own complications in terms of quality control and potential biases.

The issue of bias in data labeling is particularly concerning. Annotators bring their own cultural, personal, and professional biases to the labeling process, which can be reflected in the resulting dataset. For example, Wang et al. (2019) found that image datasets labeled predominantly by annotators from one geographic region showed biases in object recognition tasks, performing poorly on images from other regions. This highlights the need for diverse annotator pools and careful consideration of potential biases in the labeling process.

Data privacy and ethical considerations also pose challenges in data labeling. Leading data labeling companies have developed specialized solutions for these challenges. Scale AI, for instance, maintains dedicated teams and secure infrastructure for handling sensitive data in healthcare and finance. Appen implements strict data access controls and anonymization protocols, while Labelbox offers private cloud deployments for organizations with strict security requirements. When dealing with sensitive data, such as medical records or personal communications, ensuring annotator access while maintaining data privacy can be complex.

The dynamic nature of real-world data presents another limitation. Labels that are accurate at the time of annotation may become outdated or irrelevant as the underlying distribution of data changes over time. This concept, known as concept drift, necessitates ongoing labeling efforts and periodic re-evaluation of existing labels.

Lastly, the limitations of current labeling approaches become apparent when dealing with edge cases or rare events. In many real-world applications, it's the unusual or rare instances that are often most critical (e.g., rare diseases in medical diagnosis, or unusual road conditions in autonomous driving). However, these cases are, by definition, underrepresented in most datasets and may be overlooked or mislabeled in large-scale annotation efforts.

6.7.6 Case Study: KWS

The complex requirements of KWS reveal the role of automated data labeling in modern machine learning. The Multilingual Spoken Words Corpus (MSWC) (Mazumder et al. 2021) illustrates this through its innovative approach to generating labeled wake word data at scale. MSWC is large, containing over 23.4 million one-second spoken examples across 340,000 keywords in 50 different languages.

The core of this system, as illustrated in Figure 6.13, begins with paired audio recordings and corresponding transcriptions, readily accessible on various platforms like YouTube. The system processes paired audio-text inputs through forced alignment[11] to identify word boundaries, extracts individual keywords as one-second segments, and generates a large-scale multilingual dataset suitable for training keyword spotting models. For example, when a speaker says, "He gazed up the steep bank," their voice generates a complex acoustic signal that conveys more than just the words themselves. This signal encapsulates subtle transitions between words, variations in pronunciation, and the natural rhythm of speech. The primary challenge lies in accurately pinpointing the exact location of each word within this continuous audio stream.

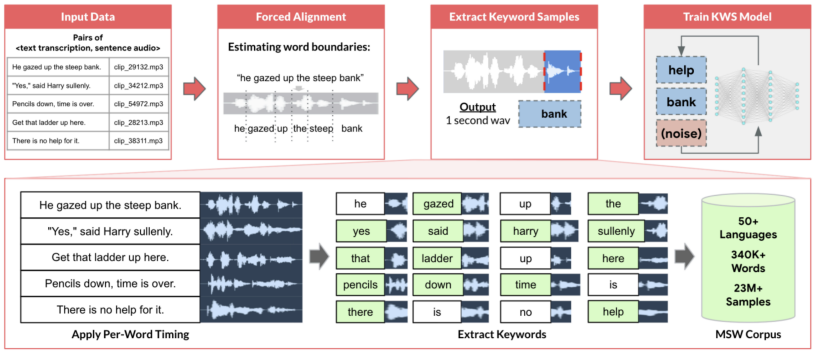


Figure 6.13: MSWC's automated data labeling pipeline.

This is where automated forced alignment proves useful. The Montreal Forced Aligner analyzes both the audio and its transcription, mapping the relationship between written words and spoken sounds. It carefully marks the boundaries of each word, accounting for how sounds blend together in natural speech. The word "bank" isn't simply at the end of the utterance – the system identifies the exact milliseconds where it begins and ends.

With these precise timestamps, the extraction system can generate clean, one-second samples of individual keywords. However, this process requires careful engineering decisions. Background noise might interfere with word boundaries. Speakers may stretch or compress words in unexpected ways. The extraction system must handle these variations while maintaining the acoustic integrity of each sample.

Quality control permeates every step of this process. The system employs self-supervised anomaly detection, using acoustic embeddings to identify potential issues. This automated validation becomes particularly crucial given

[11] **Forced Alignment:** A technique in audio processing that synchronizes spoken words in an audio file with their corresponding text transcription by analyzing phoneme-level timing.

the scale of the dataset—over 23 million samples across more than 340,000 words in 50+ languages. Therefore, traditional manual review simply couldn't maintain consistent standards across such volume.

Modern voice assistant developers often build upon this type of labeling foundation. While MSWC provides automated labeling at scale, production systems may add targeted human verification for challenging cases. Expert linguists might review edge cases, particularly for new languages or difficult acoustic environments. The infrastructure must gracefully coordinate between automated processing and human expertise.

The impact of this careful engineering extends far beyond the dataset itself. The automated pipeline has transformed how we approach wake word detection across languages. Where manual annotation might yield thousands of examples, MSWC generates millions while maintaining consistent quality. This enables voice interfaces to understand an ever-expanding vocabulary across the world's languages.

Through this approach to data labeling, MSWC demonstrates how thoughtful data engineering directly impacts production machine learning systems. The careful orchestration of forced alignment, extraction, and quality control creates a foundation for reliable voice interaction across languages and acoustic environments. When a voice assistant responds to its wake word, it draws upon this sophisticated labeling infrastructure—a testament to the power of automated data processing in modern machine learning systems.

6.8 Data Storage

Machine learning workloads have data access patterns that differ markedly from those of traditional transactional systems or routine analytics. Whereas transactional databases optimize for frequent writes and row-level updates, most ML pipelines rely on high-throughput reads, large-scale data scans, and evolving schemas. This difference reflects the iterative nature of model development: data scientists repeatedly load and transform vast datasets to engineer features, test new hypotheses, and refine models.

Additionally, ML pipelines must accommodate real-world considerations such as evolving business requirements, new data sources, and changes in data availability. These realities push storage solutions to be both scalable and flexible, ensuring that organizations can manage data collected from diverse channels—from sensor feeds to social media text—without constantly retooling the entire infrastructure. In this section, we will compare the practical use of databases, data warehouses, and data lakes for ML projects, then delve into how specialized services, metadata, and governance practices unify these varied systems into a coherent strategy.

6.8.1 Storage Systems

All raw and labeled data needs to be stored and accessed efficiently. When considering storage systems for ML, it is essential to understand the differences among different storage systems: databases, data warehouses, and data lakes. Each system has its strengths and is suited to different aspects of ML workflows.

Table Table 6.1 provides an overview of these storage systems. Databases usually support operational and transactional purposes. They work well for smaller, well-structured datasets, but can become cumbersome and expensive when applied to large-scale ML contexts involving unstructured data (such as images, audio, or free-form text).

Table 6.1: Comparative overview of the database, data warehouse, and data lake.

Attribute	Conventional Database	Data Warehouse	Data Lake
Purpose	Operational and transactional	Analytical and reporting	Storage for raw and diverse data for future processing
Data type	Structured	Structured	Structured, semi-structured, and unstructured
Scale	Small to medium volumes	Medium to large volumes	Large volumes of diverse data
Performance Optimization	Optimized for transactional queries (OLTP)	Optimized for analytical queries (OLAP)	Optimized for scalable storage and retrieval
Examples	MySQL, PostgreSQL, Oracle DB	Google BigQuery, Amazon Redshift, Microsoft Azure Synapse	Google Cloud Storage, AWS S3, Azure Data Lake Storage

Data warehouses, by contrast, are optimized for analytical queries across integrated datasets that have been transformed into a standardized schema. As indicated in the table, they handle large volumes of integrated data. Many ML systems successfully draw on data warehouses to power model training because the structured environment simplifies data exploration and feature engineering. Yet one limitation remains: a data warehouse may not accommodate truly unstructured data or rapidly changing data formats, particularly if the data originates from web scraping or Internet of Things (IoT) sensors.

Data lakes address this gap by storing structured, semi-structured, and unstructured data in its native format, deferring schema definitions until the point of reading or analysis (sometimes called *schema-on-read*)[12] . As Table Table 6.1 shows, data lakes can handle large volumes of diverse data types. This approach grants data scientists tremendous latitude when dealing with experimental use cases or novel data types. However, data lakes also demand careful cataloging and metadata management. Without sufficient governance, these expansive repositories risk devolving into unsearchable, disorganized silos.

[12] **Schema-on-read:** A data management approach where data schema definitions are applied at the time of query or analysis rather than during initial data storage.

The examples provided in Table Table 6.1 illustrate the range of technologies available for each storage system type. For instance, MySQL represents a traditional database system, while solutions like Google BigQuery and Amazon Redshift are examples of modern, cloud-based data warehouses. For data lakes, cloud storage solutions such as Google Cloud Storage, AWS S3, and Azure Data Lake Storage are commonly used due to their scalability and flexibility.

6.8.2 Storage Considerations

While traditional storage systems provide a foundation for ML workflows, the unique characteristics of machine learning workloads necessitate additional

considerations. These ML-specific storage needs stem from the nature of ML development, training, and deployment processes, and addressing them is necessary for building efficient and scalable ML systems.

One of the primary challenges in ML storage is handling large model weights. Modern ML models, especially deep learning models, can have millions or even billions of parameters. For instance, GPT-3, a large language model, has 175 billion parameters, requiring approximately 350 GB of storage just for the model weights (Brown et al. 2020). Storage systems need to be capable of handling these large, often dense, numerical arrays efficiently, both in terms of storage capacity and access speed. This requirement goes beyond traditional data storage and enters the realm of high-performance computing storage solutions.

The iterative nature of ML development introduces another critical storage consideration: versioning for both datasets and models. Unlike traditional software version control, ML versioning needs to track large binary files efficiently. As data scientists experiment with different model architectures and hyperparameters, they generate numerous versions of models and datasets. Effective storage systems for ML must provide mechanisms to track these changes, revert to previous versions, and maintain reproducibility throughout the ML lifecycle. This capability is essential not only for development efficiency but also for regulatory compliance and model auditing in production environments.

Distributed training, often necessary for large models or datasets, generates substantial intermediate data, including partial model updates, gradients, and checkpoints. Storage systems for ML need to handle frequent, possibly concurrent, read and write operations of these intermediate results. Moreover, they should provide low-latency access to support efficient synchronization between distributed workers. This requirement pushes storage systems to balance between high throughput for large data transfers and low latency for quick synchronization operations.

The diversity of data types in ML workflows presents another unique challenge. ML systems often work with a wide variety of data – from structured tabular data to unstructured images, audio, and text. Storage systems need to efficiently handle this diversity, often requiring a combination of different storage technologies optimized for specific data types. For instance, a single ML project might need to store and process tabular data in a columnar format for efficient feature extraction, while also managing large volumes of image data for computer vision tasks.

As organizations collect more data and create more sophisticated models, storage systems need to scale seamlessly. This scalability should support not just growing data volumes, but also increasing concurrent access from multiple data scientists and ML models. Cloud-based object storage systems have emerged as a popular solution due to their virtually unlimited scalability, but they introduce their own challenges in terms of data access latency and cost management.

The tension between sequential read performance for training and random access for inference is another key consideration. While training on large datasets benefits from high-throughput sequential reads, many ML serving scenarios require fast random access to individual data points or features.

Storage systems for ML need to balance these potentially conflicting requirements, often leading to tiered storage architectures where frequently accessed data is kept in high-performance storage while less frequently used data is moved to cheaper, higher-latency storage.

The choice and configuration of storage systems can significantly impact the performance, cost-effectiveness, and overall success of ML initiatives. As the field of machine learning continues to evolve, storage solutions will need to adapt to meet the changing demands of increasingly sophisticated ML workflows.

6.8.3 Performance Considerations

The performance of storage systems is critical in ML workflows, directly impacting the efficiency of model training, the responsiveness of inference, and the overall productivity of data science teams. Understanding and optimizing storage performance requires a focus on several key metrics and strategies tailored to ML workloads.

One of the primary performance metrics for ML storage is throughput, particularly for large-scale data processing and model training. High throughput is essential when ingesting and preprocessing vast datasets or when reading large batches of data during model training. For instance, distributed training of deep learning models on large datasets may require sustained read throughput of several gigabytes per second to keep GPU accelerators fully utilized.

Latency is another metric, especially for online inference and interactive data exploration. Low latency access to individual data points or small batches of data is vital for maintaining responsive ML services. In recommendation systems or real-time fraud detection, for example, storage systems must be able to retrieve relevant features or model parameters within milliseconds to meet strict service level agreements (SLAs).

The choice of file format can significantly impact both throughput and latency. Columnar storage formats such as Parquet or ORC[13] are particularly well-suited for ML workloads. These formats allow for efficient retrieval of specific features without reading entire records, substantially reducing I/O operations and speeding up data loading for model training and inference. For example, when training a model that only requires a subset of features from a large dataset, columnar formats can reduce data read times by an order of magnitude compared to row-based formats.

[13] **Parquet and ORC:** Columnar storage formats optimized for analytical workloads and machine learning pipelines. They store data by columns rather than rows, enabling selective retrieval of specific features and reducing I/O overhead for large datasets.

Compression is another key factor in storage performance optimization. While compression reduces storage costs and can improve read performance by reducing the amount of data transferred from disk, it also introduces computational overhead for decompression. The choice of compression algorithm often involves a trade-off between compression ratio and decompression speed. For ML workloads, fast decompression is usually prioritized over maximum compression, with algorithms like Snappy or LZ4 being popular choices.

Data partitioning strategies play a role in optimizing query performance for ML workloads. By intelligently partitioning data based on frequently used query parameters (such as date ranges or categorical variables), systems can

dramatically improve the efficiency of data retrieval operations. For instance, in a recommendation system processing user interactions, partitioning data by user demographic attributes and time periods can significantly speed up the retrieval of relevant training data for personalized models.

To handle the scale of data in modern ML systems, distributed storage architectures are often employed. These systems, such as HDFS (Hadoop Distributed File System) or cloud-based object stores like Amazon S3, distribute data across multiple machines or data centers. This approach not only provides scalability but also enables parallel data access, which can substantially improve read performance for large-scale data processing tasks common in ML workflows.

Caching strategies are also vital for optimizing storage performance in ML systems. In-memory caching of frequently accessed data or computed features can significantly reduce latency and computational overhead. Distributed caching systems like Redis or Memcached are often used to scale caching capabilities across clusters of machines, providing low-latency access to hot data for distributed training or serving systems.

As ML workflows increasingly span from cloud to edge devices, storage performance considerations must extend to these distributed environments. Edge caching and intelligent data synchronization strategies become needed for maintaining performance in scenarios where network connectivity may be limited or unreliable. In the end, the goal is to create a storage infrastructure that can handle the volume and velocity of data in ML workflows while providing the low-latency access needed for responsive model training and inference.

6.8.4 Storage Across ML Lifecycle Phases

The storage needs of machine learning systems evolve significantly across different phases of the ML lifecycle. Understanding these changing requirements is important for designing effective and efficient ML data infrastructures.

6.8.4.1 Development Phase

In the development phase, storage systems play a critical role in supporting exploratory data analysis and iterative model development. This stage demands flexibility and collaboration, as data scientists often work with various datasets, experiment with feature engineering techniques, and rapidly iterate on model designs to refine their approaches.

One of the key challenges at this stage is managing the versions of datasets used in experiments. While traditional version control systems like Git excel at tracking code changes, they fall short when dealing with large datasets. This gap has led to the emergence of specialized tools like DVC (Data Version Control), which enable data scientists to efficiently track dataset changes, revert to previous versions, and share large files without duplication. These tools ensure that teams can maintain reproducibility and transparency throughout the iterative development process.

Balancing data accessibility and security further complicates the storage requirements in this phase. Data scientists require seamless access to datasets

for experimentation, but organizations must simultaneously safeguard sensitive data. This tension often results in the implementation of sophisticated access control mechanisms, ensuring that datasets remain both accessible and protected. Secure data sharing systems enhance collaboration while adhering to strict organizational and regulatory requirements, enabling teams to work productively without compromising data integrity.

6.8.4.2 Training Phase

The training phase presents unique storage challenges due to the sheer volume of data processed and the computational intensity of model training. At this stage, the interplay between storage performance and computational efficiency becomes critical, as modern ML algorithms demand seamless integration between data access and processing.

To meet these demands, high-performance storage systems must provide the throughput required to feed data to multiple GPU or TPU accelerators simultaneously. Distributed training scenarios amplify this need, often requiring data transfer rates in the gigabytes per second range to ensure that accelerators remain fully utilized. This highlights the importance of optimizing storage for both capacity and speed.

Beyond data ingestion, managing intermediate results and checkpoints is another critical challenge in the training phase. Long-running training jobs frequently save intermediate model states to allow for resumption in case of interruptions. These checkpoints can grow significantly in size, especially for large-scale models, necessitating storage solutions that enable efficient saving and retrieval without impacting overall performance.

Complementing these systems is the concept of burst buffers[14] , borrowed from high-performance computing. These high-speed, temporary storage layers are particularly valuable during training, as they can absorb large, bursty I/O operations. By buffering these spikes in demand, burst buffers help smooth out performance fluctuations and reduce the load on primary storage systems, ensuring that training pipelines remain efficient and reliable.

[14] **Burst Buffers:** High-speed storage layers used to absorb large, temporary I/O demands in high-performance computing, smoothing performance during data-intensive operations.

6.8.4.3 Deployment and Serving Phase

In the deployment and serving phase, the focus shifts from high-throughput batch operations during training to low-latency, often real-time, data access. This transition highlights the need to balance conflicting requirements, where storage systems must simultaneously support responsive model serving and enable continued learning in dynamic environments.

Real-time inference demands storage solutions capable of extremely fast access to model parameters and relevant features. To achieve this, systems often rely on in-memory databases or sophisticated caching strategies, ensuring that predictions can be made within milliseconds. These requirements become even more challenging in edge deployment scenarios, where devices operate with limited storage resources and intermittent connectivity to central data stores.

Adding to this complexity is the need to manage model updates in production environments. Storage systems must facilitate smooth transitions between

model versions, ensuring minimal disruption to ongoing services. Techniques like shadow deployment, where new models run alongside existing ones for validation, allow organizations to iteratively roll out updates while monitoring their performance in real-world conditions.

6.8.4.4 Monitoring and Maintenance Phase

The monitoring and maintenance phase brings its own set of storage challenges, centered on ensuring the long-term reliability and performance of ML systems. At this stage, the focus shifts to capturing and analyzing data to monitor model behavior, detect issues, and maintain compliance with regulatory requirements.

A critical aspect of this phase is managing data drift, where the characteristics of incoming data change over time. Storage systems must efficiently capture and store incoming data along with prediction results, enabling ongoing analysis to detect and address shifts in data distributions. This ensures that models remain accurate and aligned with their intended use cases.

The sheer volume of logging and monitoring data generated by high-traffic ML services introduces questions of data retention and accessibility. Organizations must balance the need to retain historical data for analysis against the cost and complexity of storing it. Strategies such as tiered storage and compression can help manage costs while ensuring that critical data remains accessible when needed.

Regulated industries often require immutable storage to support auditing and compliance efforts. Storage systems designed for this purpose guarantee data integrity and non-repudiability, ensuring that stored data cannot be altered or deleted. Blockchain-inspired solutions and write-once-read-many (WORM) technologies are commonly employed to meet these stringent requirements.

6.8.5 Feature Stores

Feature stores are a centralized repository that stores and serves pre-computed features for machine learning models, ensuring consistency between training and inference workflows. They have emerged as a critical component in the ML infrastructure stack, addressing the unique challenges of managing and serving features for machine learning models. They act as a central repository for storing, managing, and serving machine learning features, bridging the gap between data engineering and machine learning operations.

What makes feature stores particularly interesting is their role in solving several key challenges in ML pipelines. First, they address the problem of feature consistency between training and serving environments. In traditional ML workflows, features are often computed differently in offline (training) and online (serving) environments, leading to discrepancies that can degrade model performance. Feature stores provide a single source of truth for feature definitions, ensuring consistency across all stages of the ML lifecycle.

Another fascinating aspect of feature stores is their ability to promote feature reuse across different models and teams within an organization. By centralizing feature computation and storage, feature stores can significantly reduce re-

dundant work. For instance, if multiple teams are working on different models that require similar features (e.g., customer lifetime value in a retail context), these features can be computed once and reused across projects, improving efficiency and consistency.

Feature stores also play a role in managing the temporal aspects of features. Many ML use cases require correct point-in-time feature values, especially in scenarios involving time-series data or where historical context is important. Feature stores typically offer time-travel capabilities, allowing data scientists to retrieve feature values as they were at any point in the past. This is crucial for training models on historical data and for ensuring consistency between training and serving environments.

The performance characteristics of feature stores are particularly intriguing from a storage perspective. They need to support both high-throughput batch retrieval for model training and low-latency lookups for online inference. This often leads to hybrid architectures where feature stores maintain both an offline store (optimized for batch operations) and an online store (optimized for real-time serving). Synchronization between these stores becomes a critical consideration.

Feature stores also introduce interesting challenges in terms of data freshness and update strategies. Some features may need to be updated in real-time (e.g., current user session information), while others might be updated on a daily or weekly basis (e.g., aggregated customer behavior metrics). Managing these different update frequencies and ensuring that the most up-to-date features are always available for inference can be complex.

From a storage perspective, feature stores often leverage a combination of different storage technologies to meet their diverse requirements. This might include columnar storage formats like Parquet for the offline store, in-memory databases or key-value stores for the online store, and streaming platforms like Apache Kafka for real-time feature updates.

6.8.6 Caching Strategies

Caching plays a role in optimizing the performance of ML systems, particularly in scenarios involving frequent data access or computation-intensive operations. In the context of machine learning, caching strategies extend beyond traditional web or database caching, addressing unique challenges posed by ML workflows.

One of the primary applications of caching in ML systems is in feature computation and serving. Many features used in ML models are computationally expensive to calculate, especially those involving complex aggregations or time-window operations. By caching these computed features, systems can significantly reduce latency in both training and inference scenarios. For instance, in a recommendation system, caching user embedding vectors can dramatically speed up the generation of personalized recommendations.

Caching strategies in ML systems often need to balance between memory usage and computation time. This trade-off is particularly evident in large-scale distributed training scenarios. Caching frequently accessed data shards or mini-batches in memory can significantly reduce I/O overhead, but it re-

quires careful memory management to avoid out-of-memory errors, especially when working with large datasets or models.

Another interesting application of caching in ML systems is model caching. In scenarios where multiple versions of a model are deployed (e.g., for A/B testing or gradual rollout), caching the most frequently used model versions in memory can significantly reduce inference latency. This becomes especially important in edge computing scenarios, where storage and computation resources are limited.

Caching also plays a vital role in managing intermediate results in ML pipelines. For instance, in feature engineering pipelines that involve multiple transformation steps, caching intermediate results can prevent redundant computations when rerunning pipelines with minor changes. This is particularly useful during the iterative process of model development and experimentation.

One of the challenges in implementing effective caching strategies for ML is managing cache invalidation and updates. ML systems often deal with dynamic data where feature values or model parameters may change over time. Implementing efficient cache update mechanisms that balance between data freshness and system performance is an ongoing area of research and development.

Distributed caching becomes particularly important in large-scale ML systems. Technologies like Redis or Memcached are often employed to create distributed caching layers that can serve multiple training or inference nodes. These distributed caches need to handle challenges like maintaining consistency across nodes and managing failover scenarios.

Edge caching is another fascinating area in ML systems, especially with the growing trend of edge AI. In these scenarios, caching strategies need to account for limited storage and computational resources on edge devices, as well as potentially intermittent network connectivity. Intelligent caching strategies that prioritize the most relevant data or model components for each edge device can significantly improve the performance and reliability of edge ML systems.

Lastly, the concept of semantic caching[15] is gaining traction in ML systems. Unlike traditional caching that operates on exact matches, semantic caching attempts to reuse cached results for semantically similar queries. This can be particularly useful in ML systems where slight variations in input may not significantly change the output, potentially leading to substantial performance improvements.

[15] **Semantic Caching:** A caching technique that reuses results of previous computations for semantically similar queries, reducing redundancy in data processing.

6.8.7 Access Patterns

Understanding the access patterns in ML systems is useful for designing efficient storage solutions and optimizing the overall system performance. ML workloads exhibit distinct data access patterns that often differ significantly from traditional database or analytics workloads.

One of the most prominent access patterns in ML systems is sequential reading of large datasets during model training. Unlike transactional systems that typically access small amounts of data randomly, ML training often involves reading entire datasets multiple times (epochs) in a sequential manner. This

pattern is particularly evident in deep learning tasks, where large volumes of data are fed through neural networks repeatedly. Storage systems optimized for high-throughput sequential reads, such as distributed file systems or object stores, are well-suited for this access pattern.

However, the sequential read pattern is often combined with random shuffling between epochs to prevent overfitting and improve model generalization. This introduces an interesting challenge for storage systems, as they need to efficiently support both sequential and random access patterns, often within the same training job.

In contrast to the bulk sequential reads common in training, inference workloads often require fast random access to specific data points or features. For example, a recommendation system might need to quickly retrieve user and item features for real-time personalization. This necessitates storage solutions with low-latency random read capabilities, often leading to the use of in-memory databases or caching layers.

Feature stores, which we discussed earlier, introduce their own unique access patterns. They typically need to support both high-throughput batch reads for offline training and low-latency point lookups for online inference. This dual-nature access pattern often leads to the implementation of separate offline and online storage layers, each optimized for its specific access pattern.

Time-series data, common in many ML applications such as financial forecasting or IoT analytics, presents another interesting access pattern. These workloads often involve reading contiguous blocks of time-ordered data, but may also require efficient retrieval of specific time ranges or periodic patterns. Specialized time-series databases or carefully designed partitioning schemes in general-purpose databases are often employed to optimize these access patterns.

Another important consideration is the write access pattern in ML systems. While training workloads are often read-heavy, there are scenarios that involve significant write operations. For instance, continual learning systems may frequently update model parameters, and online learning systems may need to efficiently append new training examples to existing datasets.

Understanding these diverse access patterns is helpful in designing and optimizing storage systems for ML workloads. It often leads to hybrid storage architectures that combine different technologies to address various access patterns efficiently. For example, a system might use object storage for large-scale sequential reads during training, in-memory databases for low-latency random access during inference, and specialized time-series storage for temporal data analysis.

As ML systems continue to evolve, new access patterns are likely to emerge, driving further innovation in storage technologies and architectures. The challenge lies in creating flexible, scalable storage solutions that can efficiently support the diverse and often unpredictable access patterns of modern ML workloads.

6.8.8 Case Study: KWS

During development and training, KWS systems must efficiently store and manage large collections of audio data. This includes raw audio recordings from various sources (crowd-sourced, synthetic, and real-world captures), processed features (like spectrograms or MFCCs), and model checkpoints. A typical architecture might use a data lake for raw audio files, allowing flexible storage of diverse audio formats, while processed features are stored in a more structured data warehouse for efficient access during training.

KWS systems benefit significantly from feature stores, particularly for managing pre-computed audio features. For example, commonly used spectrogram representations or audio embeddings can be computed once and stored for reuse across different experiments or model versions. The feature store must handle both batch access for training and real-time access for inference, often implementing a dual storage architecture – an offline store for training data and an online store for low-latency inference.

In production, KWS systems require careful consideration of edge storage requirements. The models must be compact enough to fit on resource-constrained devices while maintaining quick access to necessary parameters for real-time wake word detection. This often involves optimized storage formats and careful caching strategies to balance between memory usage and inference speed.

6.9 Data Governance

Data governance is a significant component in the development and deployment of ML systems. It encompasses a set of practices and policies that ensure data is accurate, secure, compliant, and ethically used throughout the ML lifecycle. As ML systems become increasingly integral to decision-making processes across various domains, the importance of robust data governance has grown significantly.

One of the central challenges of data governance is addressing the unique complexities posed by ML workflows. These workflows often involve opaque processes, such as feature engineering and model training, which can obscure how data is being used. Governance practices aim to tackle these issues by focusing on maintaining data privacy, ensuring fairness, and providing transparency in decision-making processes. These practices go beyond traditional data management to address the evolving needs of ML systems.

Security and access control form an essential aspect of data governance. Implementing measures to protect data from unauthorized access or breaches is critical in ML systems, which often deal with sensitive or proprietary information. For instance, a healthcare application may require granular access controls to ensure that only authorized personnel can view patient data. Encrypting data both at rest and in transit is another common approach to safeguarding information while enabling secure collaboration among ML teams.

Privacy protection is another key pillar of data governance. As ML models often rely on large-scale datasets, there is a risk of infringing on individual privacy rights. Techniques such as differential privacy[16] can address this con-

[16] **Differential Privacy:** A technique that preserves privacy by adding random noise to outputs, ensuring individual data points remain unidentifiable.

cern by adding carefully calibrated noise to the data. This ensures that individual identities are protected while preserving the statistical patterns necessary for model training. These techniques allow ML systems to benefit from data-driven insights without compromising ethical considerations (Dwork, n.d.), which we will learn more about in the Responsible AI chapter.

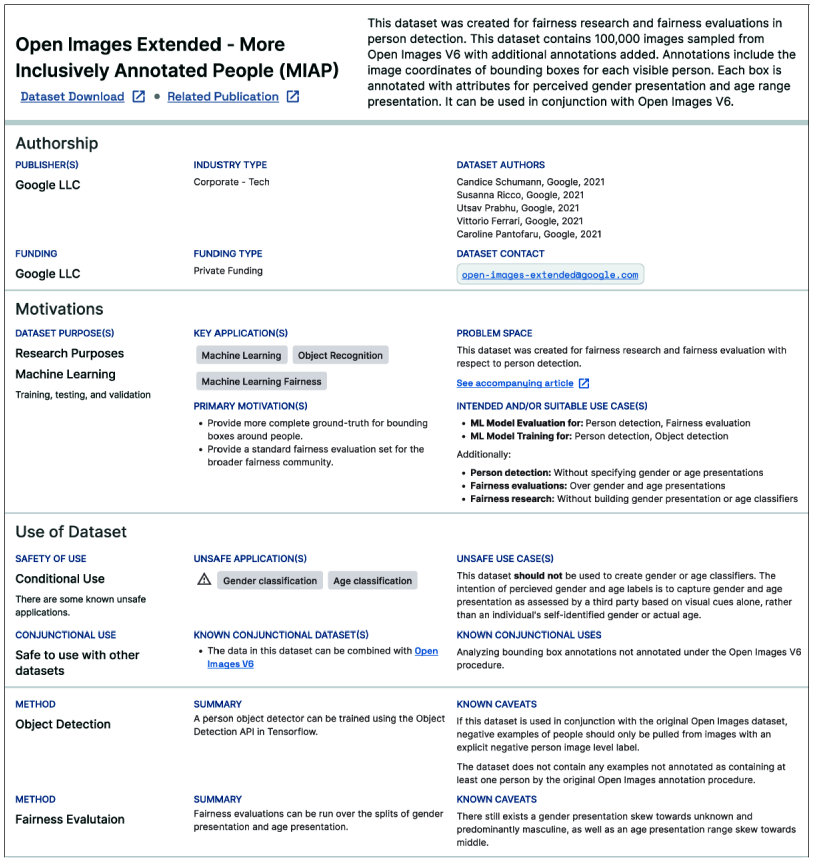
Regulatory compliance is a critical area where data governance plays a central role. Laws such as the GDPR in Europe and the HIPAA in the United States impose strict requirements on data handling. Compliance with these regulations often involves implementing features like the ability to delete data upon request or providing individuals with copies of their data. These measures not only protect individuals but also ensure organizations avoid legal and reputational risks.

Documentation and metadata management, which are often less discussed, are just as important for transparency and reproducibility in ML systems. Clear records of data lineage, including how data flows and transforms throughout the ML pipeline, are essential for accountability. Standardized documentation frameworks, such as Data Cards proposed by Pushkarna, Zaldivar, and Kjartansson (2022), offer a structured way to document the characteristics, limitations, and potential biases of datasets. For example, the Open Images Extended – More Inclusively Annotated People (MIAP) dataset uses a data card to provide detailed information about its motivations, intended use cases, and known risks. This type of documentation enables developers to evaluate datasets effectively and promotes responsible use.

Audit trails are another important component of data governance. These detailed logs track data access and usage throughout the lifecycle of ML models, from collection to deployment. Comprehensive audit trails are invaluable for troubleshooting and accountability, especially in cases of data breaches or unexpected model behavior. They help organizations understand what actions were taken and why, providing a clear path for resolving issues and ensuring compliance.

Consider a hypothetical ML system designed to predict patient outcomes in a hospital. Such a system would need to address several governance challenges. It would need to ensure that patient data is securely stored and accessed only by authorized personnel, with privacy-preserving techniques in place to protect individual identities. The system would also need to comply with healthcare regulations governing the use of patient data, including detailed documentation of how data is processed and transformed. Comprehensive audit logs would be necessary to track data usage and ensure accountability.

As ML systems grow more complex and influential, the challenges of data governance will continue to evolve. Emerging trends, such as blockchain-inspired technologies for tamper-evident logs and automated governance tools, offer promising solutions for real-time monitoring and issue detection. By adopting robust data governance practices, including tools like Data Cards, organizations can build ML systems that are transparent, ethical, and trustworthy.



Open Images Extended - More Inclusively Annotated People (MIAP)

Dataset Download • Related Publication

This dataset was created for fairness research and fairness evaluations in person detection. This dataset contains 100,000 images sampled from Open Images V6 with additional annotations added. Annotations include the image coordinates of bounding boxes for each visible person. Each box is annotated with attributes for perceived gender presentation and age range presentation. It can be used in conjunction with Open Images V6.

Authorship

PUBLISHER(S): Google LLC

INDUSTRY TYPE: Corporate - Tech

DATASET AUTHORS: Candice Schumann, Google, 2021; Susanna Ricco, Google, 2021; Utsav Prabhu, Google, 2021; Vittorio Ferrari, Google, 2021; Caroline Pantofaru, Google, 2021

FUNDING: Google LLC

FUNDING TYPE: Private Funding

DATASET CONTACT: open-images-extended@google.com

Motivations

DATASET PURPOSE(S): Research Purposes; Machine Learning; Training, testing, and validation

KEY APPLICATION(S): Machine Learning; Object Recognition; Machine Learning Fairness

PRIMARY MOTIVATION(S):
- Provide more complete ground-truth for bounding boxes around people.
- Provide a standard fairness evaluation set for the broader fairness community.

PROBLEM SPACE: This dataset was created for fairness research and fairness evaluation with respect to person detection.

See accompanying article

INTENDED AND/OR SUITABLE USE CASE(S):
- **ML Model Evaluation for:** Person detection, Fairness evaluation
- **ML Model Training for:** Person detection, Object detection

Additionally:
- **Person detection:** Without specifying gender or age presentations
- **Fairness evaluations:** Over gender and age presentations
- **Fairness research:** Without building gender presentation or age classifiers

Use of Dataset

SAFETY OF USE: Conditional Use

There are some known unsafe applications.

UNSAFE APPLICATION(S): ⚠ Gender classification; Age classification

UNSAFE USE CASE(S): This dataset **should not** be used to create gender or age classifiers. The intention of percieved gender and age labels is to capture gender and age presentation as assessed by a third party based on visual cues alone, rather than an individual's self-identified gender or actual age.

CONJUNCTIONAL USE: Safe to use with other datasets

KNOWN CONJUNCTIONAL DATASET(S):
- The data in this dataset can be combined with Open Images V6

KNOWN CONJUNCTIONAL USES: Analyzing bounding box annotations not annotated under the Open Images V6 procedure.

METHOD: Object Detection

SUMMARY: A person object detector can be trained using the Object Detection API in Tensorflow.

KNOWN CAVEATS: If this dataset is used in conjunction with the original Open Images dataset, negative examples of people should only be pulled from images with an explicit negative person image level label.

The dataset does not contain any examples not annotated as containing at least one person by the original Open Images annotation procedure.

METHOD: Fairness Evalutaion

SUMMARY: Fairness evaluations can be run over the splits of gender presentation and age presentation.

KNOWN CAVEATS: There still exists a gender presentation skew towards unknown and predominantly masculine, as well as an age presentation range skew towards middle.

Figure 6.14: Data card example for the Open Images Extended dataset.

6.10 Conclusion

Data engineering is the backbone of any successful ML system. By thoughtfully defining problems, designing robust pipelines, and practicing rigorous data governance, teams establish a foundation that directly influences model performance, reliability, and ethical standing. Effective data acquisition strategies—whether through existing datasets, web scraping, or crowdsourcing—must balance the realities of domain constraints, privacy obligations, and labeling complexities. Likewise, decisions around data ingestion (batch or streaming) and transformation (ETL or ELT) affect both cost and throughput, with monitoring and observability essential to detect shifting data quality.

Throughout this chapter, we saw how critical it is to prepare data well in advance of modeling. Data labeling emerges as a particularly delicate phase: it involves human effort, requires strong quality control practices, and has ethical ramifications. Storage choices—relational databases, data warehouses, data lakes, or specialized systems—must align with both the volume and velocity of

ML workloads. Feature stores and caching strategies support efficient retrieval across training and serving pipelines, while good data governance ensures adherence to legal regulations, protects privacy, and maintains stakeholder trust.

All these elements interlock to create an ecosystem that reliably supplies ML models with the high-quality data they need. When done well, data engineering empowers teams to iterate faster, confidently deploy new features, and build systems capable of adapting to real-world complexity. The next chapters will build on these foundations, exploring how optimized training, robust model operations, and security considerations together form a holistic approach to delivering AI solutions that perform reliably and responsibly at scale.

6.11 Resources

Here is a curated list of resources to support students and instructors in their learning and teaching journeys. We are continuously working on expanding this collection and will add new exercises soon.

Slides

These slides are a valuable tool for instructors to deliver lectures and for students to review the material at their own pace. We encourage students and instructors to leverage these slides to improve their understanding and facilitate effective knowledge transfer.

- Data Engineering: Overview.
- Feature engineering.
- Data Standards: Speech Commands.
- Crowdsourcing Data for the Long Tail.
- Reusing and Adapting Existing Datasets.
- Responsible Data Collection.
- Data Anomaly Detection:
  - Anomaly Detection: Overview.
  - Anomaly Detection: Challenges.
  - Anomaly Detection: Datasets.
  - Anomaly Detection: using Autoencoders.

Videos

- *Coming soon.*

Exercises

To reinforce the concepts covered in this chapter, we have curated a set of exercises that challenge students to apply their knowledge and deepen their understanding.

- Exercise 1
- Exercise 2
- Exercise 3

References

Akidau, Tyler, Robert Bradshaw, Craig Chambers, Slava Chernyak, Rafael J. Fernández-Moctezuma, Reuven Lax, Sam McVeety, et al. 2015. "The Dataflow Model: A Practical Approach to Balancing Correctness, Latency, and Cost in Massive-Scale, Unbounded, Out-of-Order Data Processing." *Proceedings of the VLDB Endowment* 8 (12): 1792–1803. https://doi.org/10.14778/2824032.2824076.

Ardila, Rosana, Megan Branson, Kelly Davis, Michael Kohler, Josh Meyer, Michael Henretty, Reuben Morais, Lindsay Saunders, Francis Tyers, and Gregor Weber. 2020. "Common Voice: A Massively-Multilingual Speech Corpus." In *Proceedings of the Twelfth Language Resources and Evaluation Conference*, 4218–22. Marseille, France: European Language Resources Association. https://aclanthology.org/2020.lrec-1.520.

Bishop, Christopher M. 2006. *Pattern Recognition and Machine Learning*. Springer.

Blackwood, Jayden, Frances C. Wright, Nicole J. Look Hong, and Anna R. Gagliardi. 2019. "Quality of DCIS Information on the Internet: A Content Analysis." *Breast Cancer Research and Treatment* 177 (2): 295–305. https://doi.org/10.1007/s10549-019-05315-8.

Brown, Tom B., Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, et al. 2020. "Language Models Are Few-Shot Learners." *Advances in Neural Information Processing Systems* 33: 1877–1901.

Chapelle, O., B. Scholkopf, and A. Zien Eds. 2009. "Semi-Supervised Learning (Chapelle, o. Et Al., Eds.; 2006) [Book Reviews]." *IEEE Transactions on Neural Networks* 20 (3): 542–42. https://doi.org/10.1109/tnn.2009.2015974.

Dwork, Cynthia. n.d. "Differential Privacy: A Survey of Results." In *Theory and Applications of Models of Computation*, 1–19. Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-540-79228-4/_1.

Gebru, Timnit, Jamie Morgenstern, Briana Vecchione, Jennifer Wortman Vaughan, Hanna Wallach, Hal Daumé III, and Kate Crawford. 2021. "Datasheets for Datasets." *Communications of the ACM* 64 (12): 86–92. https://doi.org/10.1145/3458723.

Gudivada, Venkat N., Dhana Rao Rao, et al. 2017. "Data Quality Considerations for Big Data and Machine Learning: Going Beyond Data Cleaning and Transformations." *IEEE Transactions on Knowledge and Data Engineering*.

Inmon, W. H. 2005. *Building the Data Warehouse*. John Wiley & Sons.

Johnson-Roberson, Matthew, Charles Barto, Rounak Mehta, Sharath Nittur Sridhar, Karl Rosaen, and Ram Vasudevan. 2017. "Driving in the Matrix: Can Virtual Worlds Replace Human-Generated Annotations for Real World Tasks?" In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, 746–53. Singapore, Singapore: IEEE. https://doi.org/10.1109/icra.2017.7989092.

Kleppmann, Martin. 2016. *Designing Data-Intensive Applications: The Big Ideas Behind Reliable, Scalable, and Maintainable Systems*. O'Reilly Media. http://shop.oreilly.com/product/0636920032175.do.

Krishnan, Rayan, Pranav Rajpurkar, and Eric J. Topol. 2022. "Self-Supervised Learning in Medicine and Healthcare." *Nature Biomedical Engineering* 6 (12): 1346–52. https://doi.org/10.1038/s41551-022-00914-1.

Kuhn, Max, and Kjell Johnson. 2013. *Applied Predictive Modeling*. Springer New York. https://doi.org/10.1007/978-1-4614-6849-3.

Mazumder, Mark, Sharad Chitlangia, Colby Banbury, Yiping Kang, Juan Manuel Ciro, Keith Achorn, Daniel Galvez, et al. 2021. "Multilingual Spoken Words Corpus." In *Thirty-Fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 2)*.

Northcutt, Curtis G, Anish Athalye, and Jonas Mueller. 2021. "Pervasive Label Errors in Test Sets Destabilize Machine Learning Benchmarks." *arXiv*. https://doi.org/https://doi.org/10.48550/arXiv.2103.14749 arXiv-issued DOI via DataCite.

Oakden-Rayner, Luke, Jared Dunnmon, Gustavo Carneiro, and Christopher Re. 2020. "Hidden Stratification Causes Clinically Meaningful Failures in Machine Learning for Medical Imaging." In *Proceedings of the ACM Conference on Health, Inference, and Learning*, 151–59. ACM. https://doi.org/10.1145/3368555.3384468.

Pineau, Joelle, Philippe Vincent-Lamarre, Koustuv Sinha, Vincent Larivière, Alina Beygelzimer, Florence d'Alché-Buc, Emily Fox, and Hugo Larochelle. 2021. "Improving Reproducibility in Machine Learning Research (a Report from the Neurips 2019 Reproducibility Program)." *Journal of Machine Learning Research* 22 (164): 1–20.

Pushkarna, Mahima, Andrew Zaldivar, and Oddur Kjartansson. 2022. "Data Cards: Purposeful and Transparent Dataset Documentation for Responsible AI." In *2022 ACM Conference on Fairness, Accountability, and Transparency*, 1776–826. ACM. https://doi.org/10.1145/3531146.3533231.

Ratner, Alex, Braden Hancock, Jared Dunnmon, Roger Goldman, and Christopher Ré. 2018. "Snorkel MeTaL: Weak Supervision for Multi-Task Learning." In *Proceedings of the Second Workshop on Data Management for End-to-End Machine Learning*. ACM. https://doi.org/10.1145/3209889.3209898.

Sambasivan, Nithya, Shivani Kapania, Hannah Highfill, Diana Akrong, Praveen Paritosh, and Lora M Aroyo. 2021. "'Everyone Wants to Do the Model Work, Not the Data Work': Data Cascades in High-Stakes AI." In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*, 1–15. ACM. https://doi.org/10.1145/3411764.3445518.

Sculley, David, Gary Holt, Daniel Golovin, Eugene Davydov, Todd Phillips, Dietmar Ebner, Vinay Chaudhary, Michael Young, Jean-Francois Crespo, and Dan Dennison. 2021. "Technical Debt in Machine Learning Systems."

In *Technical Debt in Practice*, 177–92. The MIT Press. https://doi.org/10.7551/mitpress/12440.003.0011.

Sheng, Victor S., and Jing Zhang. 2019. "Machine Learning with Crowdsourcing: A Brief Summary of the Past Research and Future Directions." *Proceedings of the AAAI Conference on Artificial Intelligence* 33 (01): 9837–43. https://doi.org/10.1609/aaai.v33i01.33019837.

Strickland, Eliza. 2019. "IBM Watson, Heal Thyself: How IBM Overpromised and Underdelivered on AI Health Care." *IEEE Spectrum* 56 (4): 24–31. https://doi.org/10.1109/mspec.2019.8678513.

Wang, Tianlu, Jieyu Zhao, Mark Yatskar, Kai-Wei Chang, and Vicente Ordonez. 2019. "Balanced Datasets Are Not Enough: Estimating and Mitigating Gender Bias in Deep Image Representations." In *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, 5309–18. IEEE. https://doi.org/10.1109/iccv.2019.00541.

Warden, Pete. 2018. "Speech Commands: A Dataset for Limited-Vocabulary Speech Recognition." *arXiv Preprint arXiv:1804.03209*, April. http://arxiv.org/abs/1804.03209v1.