Summarize

Chapter 8. Dataset Engineering

The quality of a model depends on the quality of its training data. The best ML team in the world with infinite compute can't help you finetune a good model if you don't have data. The goal of dataset engineering is to create a dataset that allows you to train the best model, ideally within your allocated budget.

As fewer companies can afford to develop models from scratch, more are turning to data to differentiate their AI performance. As models demand more data, data handling becomes more challenging and demands more investments in talent and infrastructure.[1]

Data operations have evolved from side tasks that people handle when they have time to dedicated roles. Many AI companies now employ data labelers, dataset creators, and data quality engineers, either integrated into or working alongside their core engineering teams.

If the model landscape is confusing enough with numerous offerings, the data landscape is even more complex, with an ever-growing array of datasets and techniques being introduced. This chapter gives you an overview of the data landscape and considerations to take into account when building your own dataset.

It begins with data curation, addressing questions like What data do you need? How much? What does it mean for data to be of high quality? It then discusses techniques for data synthesis and processing. Data curation, generation, and processing don't follow a linear path. You'll likely have to go back and forth between different steps.

For the same model, different training phases aim to teach the model different capabilities, and, therefore, require datasets with different attributes. For example, data quantity for pre-training is often measured in the number of tokens, whereas data quantity for supervised finetuning is often measured in the number of examples. However, at a high level, their curation processes follow the same principle. This chapter focuses

on post-training data because that's more relevant to application developers. However, I'll also include lessons from pre-training data when these lessons are insightful for post-training.

There are best practices you can follow and tools that you can use to automate parts of the process. However, data will mostly just be toil, tears, and sweat.

A DATA-CENTRIC VIEW OF AI

The increasing focus on data during AI development has given rise to *data-centric AI*, as opposed to *model-centric AI:*

- Model-centric AI tries to improve AI performance by enhancing the models themselves. This involves designing new architectures, increasing the sizes of the models, or developing new training techniques.
- Data-centric AI tries to improve AI performance by enhancing the data. This involves developing new data processing techniques and creating high-quality datasets that allow better models to be trained with fewer resources.

In the early days of deep learning, many AI benchmarks were model-centric. Given a dataset like ImageNet, people try to train the best possible model using the same dataset. In recent years, more benchmarks have become data-centric. Given the same model, people try to develop a dataset that gives this model the best performance.

In 2021, Andrew Ng launched a data-centric AI competition where participants needed to improve upon the same base dataset by applying techniques such as fixing incorrect labels, adding edge case examples, augmenting data, etc.

In 2023, DataComp (Gadre et al., 2023) hosted a competition whose goal was to create the best dataset for training a CLIP model (Radford et al., 2021). A standardized script trains a CLIP model on each submitted dataset. The quality of a dataset is evaluated based on its resulting model's performance on 38 downstream tasks. In 2024, they hosted a similar competition to evaluate datasets for language models with scales from 412M to 7B parameters (Li et al., 2024). Other similar data-centric benchmarks include DataPerf (MLCommons, 2023) and dcbench (Eyuboglu and Karlaš, 2022).

The model-centric and data-centric division helps guide research. In reality, however, meaningful technological progress often requires investment in both model and data improvements.

Data Curation

While not all issues with AI models can be solved with data, data is often a key part of the solution. The right data can make the model more capa-

ble, safer, and able to handle longer contexts. Conversely, poor data can cause the model to increase biases and hallucinations. Mistakes in data can harm the model and waste resources.

Data curation is a science that requires understanding how the model learns and what resources are available to help it learn. Dataset builders should work closely with application and model developers. In a small team, they might be the same person—the person responsible for training a model is also responsible for acquiring the data for it. However, organizations with high data demands often employ specialized roles.[2]

What data you need depends on your task and what you want to teach the model. For self-supervised finetuning, you need sequences of data. For instruction finetuning, you need data in the (instruction, response) format. For preference finetuning, you need data in the (instruction, winning response, losing response) format. To train a reward model, you can use the same data format as preference finetuning or use data with annotated scores for each of your examples in the ((instruction, response), score) format.

Training data should exhibit the behaviors you want your model to learn. Acquiring high-quality data annotations is always challenging, but it's even more challenging if you want to teach models complex behaviors such as chain-of-thought (CoT) reasoning and tool use. Let's go over these two examples to understand why:

Chain-of-thought

As discussed in Chapter 5, CoT prompting nudges the model to work through a problem step-by-step before producing the final answer. To teach a model to generate step-by-step responses, its training data should include CoT responses. "Scaling Instruction-Finetuned Language Models" (Chun et al., 2024) shows that incorporating step-by-step responses in the finetuning data greatly enhances the performance of models of various sizes on CoT tasks, with accuracy nearly doubling for certain tasks.

Generating multi-step responses can be tedious and time-consuming—explaining how to solve a math problem step-by-step is much more challenging than simply giving the final answer. To illustrate this, here are two examples, one with only the final answer and one with CoT. Both are from Chun et al. (2024):

```
Instruction: Please answer the following question. What is the boiling po
of Nitrogen?
Response (without CoT): -320.4F

CoT instruction: Answer the following question by reasoning step-by-step.
The cafeteria had 23 apples. If they used 20 for lunch and bought 6 more,
how many apples do they have?
Response (with CoT): The cafeteria had 23 apples originally. They used 20
to make lunch. So they had 23 - 20 = 3. They bought 6 more apples, so the
have 3 + 6 = 9.
```

As a result, CoT datasets are less common compared to other instruction datasets.

Tool use

Given the vast amount of knowledge a model acquires during pretraining, many models might intuitively know how to use certain tools. However, a model's tool use ability can be improved by showing it tool use examples. It's common to use domain experts to create tool use data, where each prompt is a task that requires tool use, and its response is the actions needed to perform that task. For example, if you want data to finetune a model to act as a personal assistant, you might want to ask professional personal assistants what types of tasks they usually perform, how they perform them, and what tools they need. If you ask human experts to explain how they do things, they might miss certain steps, either because of faulty memory or because they might think these steps aren't important. It's often necessary to observe how humans perform these tasks to ensure accuracy.

However, what's efficient for humans might not be efficient for AI, and vice versa. As a result, human annotations might not be ideal for AI agents. For example, a human might prefer a web interface, whereas it's easier for a model to use an API. To search for something, a human might first open a browser, copy and paste that query into the search bar, and click on each result. Meanwhile, a model can just send a request to the search API with the query and process all the results at once. For this reason, many rely on simulations and other synthetic techniques to generate tool use data, as explored later in this chapter.

Tool use data might also require special formats. In typical conversation data, the user and AI take turns, with each turn containing

one message. However, for tool use, the AI might need to generate multiple messages each turn, with each message sent to a different location. For example, it might send one message to the code interpreter and one message to the user (such as to inform the user what it's doing). To support this, Llama 3 authors (Dubey et al., 2024) designed a multi-message chat format that consists of message headers that specify the source and destination of each message, and special termination tokens to specify where the human and AI turns start.

When curating data for applications with conversation interfaces, you need to consider whether you require single-turn data, multi-turn data, or both. Single-turn data helps train a model to respond to individual instructions. Multi-turn data, on the other hand, teaches the model how to solve tasks—many real-world tasks involve back-and-forth. For instance, when given a query, a model may need to first clarify the user's intent before addressing the task. After the model's response, the user might provide corrections or additional information for the next step.

Single-turn data is simpler and, therefore, easier to obtain. Multi-turn data often requires purpose-built scenarios or more involved interactions to capture.

Data curation isn't just about creating new data to help a model learn new behaviors but is also about removing existing data to help a model unlearn bad behaviors. Imagine you work on a chatbot like ChatGPT and you hear user complaints that the chatbot is a bit arrogant, annoying users and wasting their tokens. For example, when a user asks it to verify if a statement is factually correct, the chatbot responds with: "The statement is correct, but its style can be improved to be better." It then continues to produce an unsolicited rewriting of the statement.

You investigate and find that in the training data, there are several examples of annotations with unsolicited suggestions. You put in a request to remove these examples from the training data and another request to acquire new examples that demonstrate fact-checking without unsolicited rewriting.

Each application might require data of different characteristics. Different training phases also require different data mixes. At a high level, however, data curation follows the three criteria: data quality, data coverage, and data quantity.

To give an intuition about these terms, if you think of model training as cooking, the data fed into the model is the ingredients. Data quality is equivalent to the quality of the ingredients—you can't have good food if your ingredients are spoiled. Data coverage is equivalent to having the right mix of ingredients (e.g., you shouldn't have too much or too little sugar). Data quantity is about how many ingredients you should have. Let's explore these terms in detail.

Data Quality

A small amount of high-quality data can outperform a large amount of noisy data, e.g., data that is irrelevant or inconsistent. The creators of the Yi model family found that 10K carefully crafted instructions are superior to hundreds of thousands of noisy instructions (Young et al., 2024).

Similarly, "LIMA: Less Is More for Alignment" (Zhou et al., 2023) shows that a 65B-parameter Llama model, finetuned with 1,000 carefully curated prompts and responses, can produce answers that are either equivalent or strictly preferred to GPT-4 in 43% of cases, as judged by human annotators. However, the downside of having too few data examples is that LIMA is not as robust as product-grade models.

The Llama 3 team also arrived at the same conclusion. Notably, they found that human-generated data is more prone to errors and inconsistencies, particularly for nuanced safety policies. This led them to develop AI-assisted annotation tools to ensure high data quality.

Most people understand the importance of data quality, but what does it mean for data to be high-quality? The short answer is that data is considered high-quality if it helps you do your job efficiently and reliably. The long answers, however, differ for different people.[3] In general, data can be considered high-quality if it has the following six characteristics: relevant, aligned with task requirements, consistent, correctly formatted, unique, and compliant. Some specific use cases might have other requirements:

Relevant

> The training examples should be relevant to the task you're training the model to do. For example, if the task is to answer legal questions today, a legal dataset from the 19th century might not be relevant. However, if the task is about the legal system in the 19th century, this dataset is highly relevant.

Aligned with task requirements

The annotations should align with the task's requirements. For example, if the task requires factual consistency, the annotations should be factually correct. If the task requires creativity, the annotations should be creative. If the task demands not just a score but also a justification for that score, the annotations should include both scores and justifications. But if the task demands concise answers, the annotations should be concise.

I used "aligned" instead of "accurate" or "correct" because, depending on the task, an accurate or correct response might not be what a user wants.

Consistent

Annotations should be consistent across examples and annotators. If you ask two annotators to annotate the same example, their annotations shouldn't be too different. If the task is to score essays from 1 to 5, would two essays with the same score be of the same quality? Inconsistent annotations can confuse the model, making it harder for the model to learn.

Having a good annotation guideline is essential for having annotations that are both aligned with task requirements and consistent.

Correctly formatted

All examples should follow the format expected by the model. Redundant formatting tokens can interfere with the model's learning, and, therefore, they should be removed. For example, if you scrape product reviews from a website, you should remove HTML tags. Beware of trailing white spaces, new lines, inconsistent casing, and numerical formats.[4]

Sufficiently unique

This refers to unique examples in your data.[5] In the context of model training, duplications can introduce biases and cause data contamination. I use "sufficiently unique" because specific use cases can tolerate different levels of duplications.

Compliant

Data should be compliant with all relevant internal and external policies (including laws and regulations). For example, if you're not allowed to use PII data to train your models, your data shouldn't contain any PII data.

Before setting out to create data, it's important to think about what each of these characteristics means for you. The techniques discussed in this section aim to produce data with these characteristics.

Data Coverage

A model's training data should cover the range of problems you expect it to solve. Real-world users often have a wide range of problems, and the way they express those problems can vary significantly. Having data that captures the diverse usage patterns of your application is key for the model to perform well. Coverage requires sufficient *data diversity*, which is why many refer to this attribute as data diversity.

For example, if some users construct detailed instructions with abundant references while some other users prefer short instructions, your finetuning data should include both detailed and short instructions. If user queries typically have typos, you should include examples with typos. If your application works with multiple programming languages, your training data should include the programming languages your users care about.

Different applications have different dimensions of diversity. For example, a French-to-English tool doesn't need language diversity but might benefit from diversity in topics, lengths, and speaking styles. On the other hand, a chatbot that recommends products to global customers doesn't necessarily need domain diversity, but linguistic and cultural diversity will be important.

For general-purpose use cases like chatbots, the finetuning data should be diverse, representing a wide range of topics and speaking patterns. Ding et al., (2023) believe that the most straightforward way to further improve the performance of chat language models is to increase the quality and diversity of data employed in the training process. To develop Nemotron (Adler et al., 2024), NVIDIA researchers focused on creating a dataset with task diversity, topic diversity, and instruction diversity, which includes instructions for different output formats, instructions with different output lengths, and instructions for open-ended answers as well as yes-or-no answers. "The Data Addition Dilemma" (Shen et al., 2024) demonstrated that in some cases, adding more heterogeneous data can lead to worse performance.

Meta shared that Llama 3 doesn't deviate significantly from older Llama versions in terms of model architecture. Llama 3's performance gains are

"primarily driven by improvements in data quality and diversity as well as by increased training scale." The Llama 3 paper has rich details on data coverage through all three phases of training: pre-training, supervised finetuning, and preference finetuning. While this chapter focuses on post-training data, it's useful to look at the *data mix* for the same model across all different training phases to compare and highlight the considerations for each phase.

A diversity axis that is consistent in all three phases is domain diversity, though what exactly *diverse* means differs, as shown in Table 8-1. This table shows only high-level domains and doesn't include finer-grained topics, like "geometry", which is a sub-category in math. Post-training data also has different diversity axes not shown in the table, such as the number of tokens (both for context and response) and the number of turns. Llama 3 uses synthetic data for post-training, so another dimension is the ratio of human-generated data to AI-generated data.

Table 8-1. For Llama 3, different training phases have different optimal domain mixes.

	Pre-training	**Supervised finetuning**	**Preference finetuning**
General knowledge (English)	50%	52.66%	81.99%
Math and reasoning	25%	21.19%	5.89%
Coding	17%	14.89%	6.93%
Multilingual	8%	3.01%	5.19%
Exam-like	X	8.14%	X
Long context	X	0.11%	X

It's interesting to note that during pre-training and supervised finetuning, the number of combined math, reasoning, and code tokens accounts for almost half of the training data. While I don't know exactly what percentage of the internet data is math and code, I believe that it's far below 50%. Llama 3 authors shared that *annealing* the model on small amounts of high-quality code and math data (training the model using an increasingly smaller learning rate with increasingly more code and math data) can boost the performance of their models on key benchmarks. This confirms a common belief that high-quality code and math data is more effective than natural language text in boosting the model's reasoning capabilities.

The percentage of code and math data during preference finetuning is much smaller (12.82% combined), likely because the goal is to reflect the real distribution of user preferences.

This brings up a question: How do we decide on the right data mix? A simple approach is to choose a data mix that accurately reflects the real-world application usage. You can also use experiments to find optimal data mixes. For example, Meta performed scaling law experiments similar to what is discussed in "Scaling extrapolation". For each candidate data mix, they trained several small models on a data mix and used that to predict the performance of a large model on that mix. The final model mix is the best-guess mix derived from the experiment results.

To evaluate the impact of data diversity and quality, Zhou et al. (2023) carried out an interesting experiment where they trained a 7B-parameter language model on three datasets of the same size—2,000 examples—but with different characteristics. The first is high-quality but not diverse. The second is diverse but low-quality. The third is both diverse and high-quality. Figure 8-1 shows the generation quality of the three resulting models.

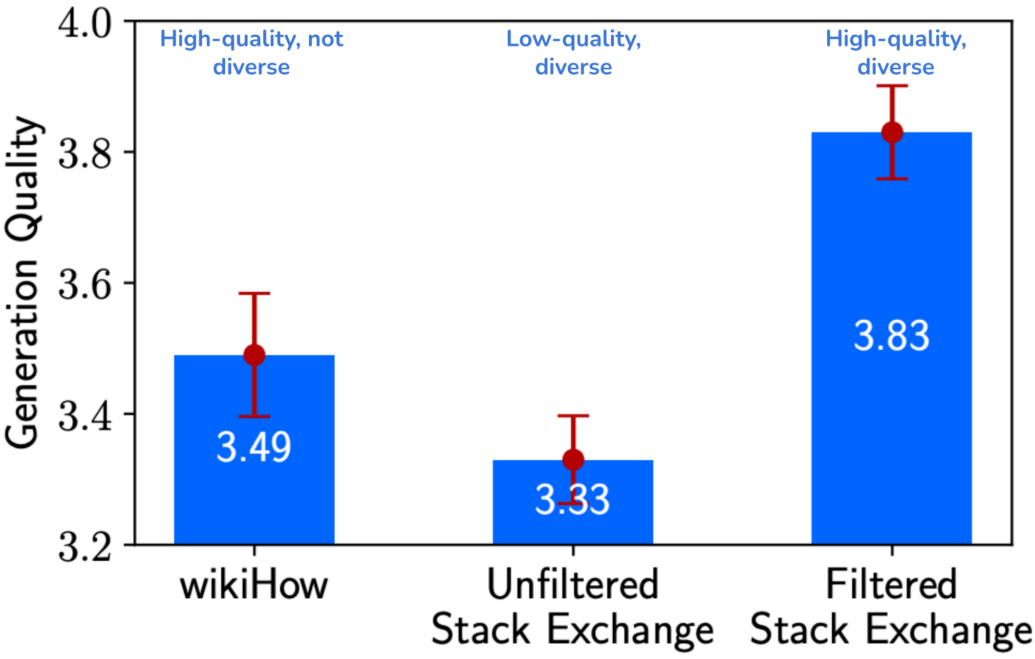


Figure 8-1. A 7B-parameter model, finetuned on a dataset that is both high-quality and diverse, outperforms that same model finetuned on a dataset that is either diverse or high-quality. Image from Zhou et al. (2023). The image is licensed under CC BY 4.0.

Data Quantity

Asking how much data you need is like asking how much money you need. The answer varies widely from one situation to the next. At one extreme, Jeremy Howard and Jonathan Whitaker did a fun experiment to show that LLMs can learn from a single example. At another extreme, some teams have finetuned models with millions of examples.

While millions of examples sounds like a lot, it's small compared to the data typically needed to train a foundation model from scratch. For reference, Llama 2 and Llama 3 were trained using 2 trillion and 16 trillion tokens, respectively. If each example is 2,000 tokens, it'd be equivalent to 1 billion and 15 billion examples.

NOTE

You might wonder: if I have millions of examples, shouldn't I just train a model from scratch? You can and should evaluate whether training a model from scratch would improve your performance. While finetuning on top of a pre-trained model is typically more efficient than training from scratch, there are situations when finetuning can be worse, especially when you have a lot of training data. This is due to a phenomenon called *ossification*, where pre-training can *ossify* (i.e., freeze) the model weights so that they don't adapt as well to the finetuning data (Hernandez et al., 2021). Smaller models are more susceptible to ossification than larger models.

Other than data quality and data diversity, three other factors influence how much data you need:

Finetuning techniques

Full finetuning promises to give the best performance, but it requires orders of magnitude more data than PEFT methods like LoRA. If you have tens of thousands to millions of (instruction, response) pairs, you might want to attempt full finetuning. If you have only a few hundred or a few thousand examples, PEFT might work best.

Task complexity

A simple task, such as classifying whether a product review is positive or negative, will require much less data than a complex task, such as a question answering about financial filings.

Base model's performance

The closer the base model is to the desirable performance, the fewer examples are needed to get there. Assuming that bigger base models are better, you might need fewer examples to finetune big models. This is the opposite of pre-training, where bigger models need more training data.

OpenAI's finetuning guide shows that if you have fewer examples (100), more advanced models give you better finetuning performance. This is

likely because the more advanced models already perform better out of the box. However, after finetuning on a lot of examples (550,000), all five models in the experiment performed similarly, as illustrated in Figure 8-2.

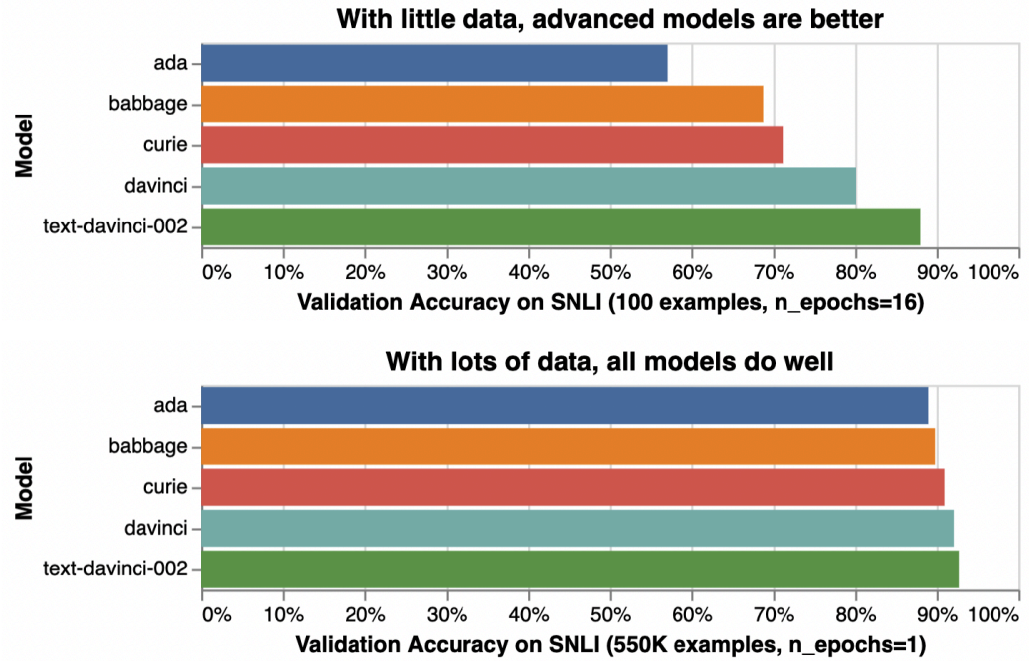


Figure 8-2. With 100 examples, more advanced models give much better performance after finetuning. With 550,000 examples, all models give similar performance after finetuning. Experiments done by Stanford Natural Language Inference (SNLI) Corpus.

In short, if you have a small amount of data, you might want to use PEFT methods on more advanced models. If you have a large amount of data, use full finetuning with smaller models.

Before investing in curating a large dataset, you might want to start with a small, well-crafted dataset (e.g., 50 examples) to see if finetuning can improve the model. If this small dataset is sufficient to achieve your desirable performance, that's great. Clear improvements suggest that more data will improve the performance even more. If no improvement is observed with small data, a bigger dataset will rarely do the trick.

However, be careful before concluding that finetuning with a small dataset doesn't improve a model. Many things, other than data, can impact finetuning's results, such as the choice of hyperparameters (e.g., the learning rate is too high or too low), data quality, poorly crafted prompts, etc. *In the vast majority of cases, you should see improvements after finetuning with 50–100 examples.*

TIP

It's possible to reduce the amount of high-quality data needed by first finetuning your model using lower-quality or less-relevant data. Here are three examples of this approach:

Self-supervised → supervised

You want to finetune a model to answer legal questions. Your (question, answer) set is small, but you have many legal documents. You can first finetune your model on legal documents in a self-supervised manner, then further finetune the model on (question, answer) pairs.

Less-relevant data → relevant data

You want to finetune a model to classify sentiments for product reviews, but you have little product sentiment data and much more tweet sentiment data. You can first finetune your model to classify tweet sentiments, then further finetune it to classify product sentiments.

Synthetic data → real data

You want to finetune a model to predict medical conditions from medical reports. Due to the sensitive nature of this task, your data is limited. You can use AI models to synthesize a large amount of data to finetune your model first, then further finetune it on your real data. This approach is harder to get right, as you'll have to do two distinct finetuning jobs while coordinating the transitioning between them. If you don't know what you're doing, you might end up using more compute just to produce a model worse than what you would've gotten by just finetuning with high-quality data.[6]

Experimenting with a small dataset can help you estimate how much more data you'll need. You can finetune a model on subsets of your current dataset—e.g., 25%, 50%, 100%—and plot how performance scales with dataset size. A steep performance gain slope with increasing dataset size means that you can expect significant performance improvement by doubling your data. A plateau slope means that doubling your data will give only a small improvement. Figure 8-3 shows an example of this plot.

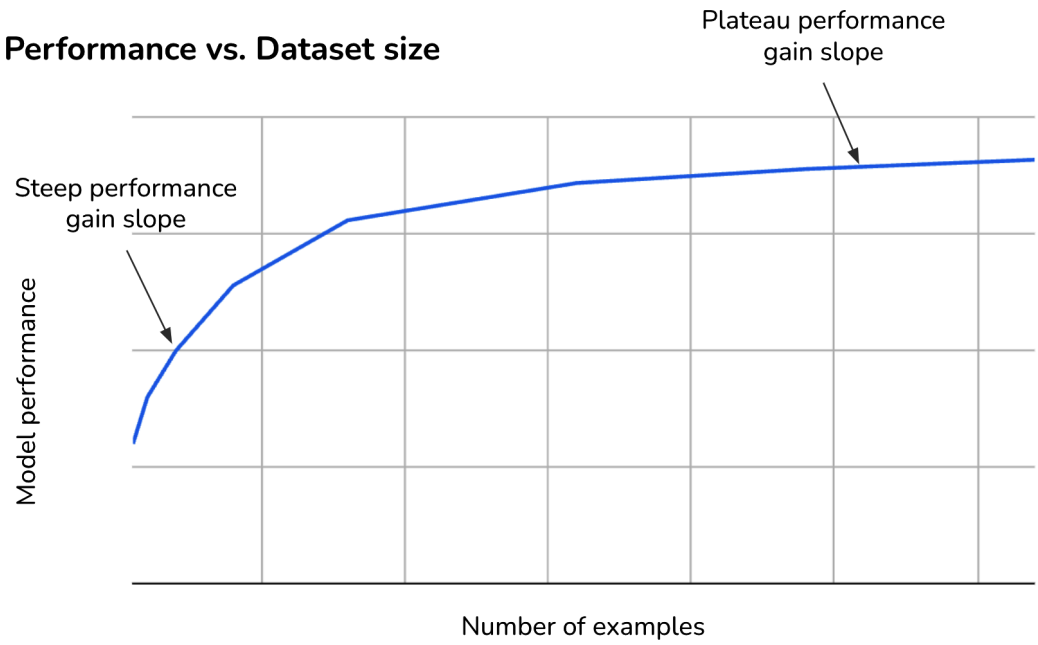


Figure 8-3. The performance gain curve with different dataset sizes can help you estimate the impact of additional training examples on your model's performance.

The performance gain curve shown in Figure 8-3 is fairly typical. In most cases, additional training examples yield diminishing returns: the same number of examples typically gives a lower performance boost as the dataset grows. For example, the first 1,000 examples might improve a model's accuracy by ten percentage points, but the next 1,000 examples might only improve it by five.

While a larger number of finetuning examples generally improves a model's performance, the diversity of the examples matters, too. The paper "Scaling Instruction-Finetuned Language Models" (Chung et al., 2022) shows that model performance increased significantly when the number of finetuning tasks increased from 9 to 282. Beyond 282 tasks, the performance gains started to plateau, though there were still positive but incremental improvements up to 1,836 tasks, as shown in Figure 8-4. This suggests that the model benefits greatly from exposure to a diverse set of tasks during finetuning.

The diversity of data can be reflected in task types (such as summarization and question answering), topic diversity (such as fashion, finance, and technology), and the expected output formats (such as JSON outputs or yes-or-no answers).

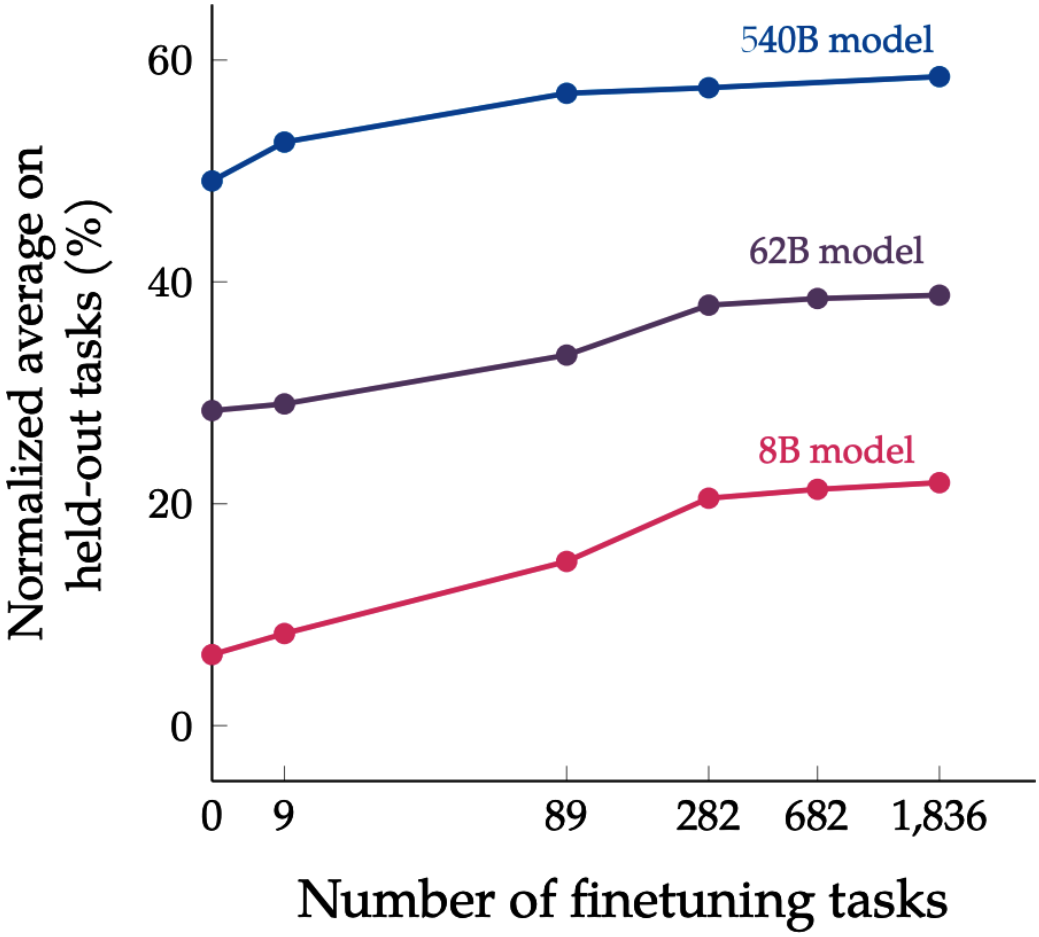


Figure 8-4. Diversity in finetuning number, measured by the number of tasks, can impact model performance. Image from "Scaling Instruction-Finetuned Language Models" (Chung et al., 2022). The image is licensed under CC BY 4.0.

How much data to use for finetuning is determined not just by what you need but also by what you can afford. If you budget $10,000 for data annotation and each example costs $2 to annotate, you can have at most 5,000 examples. You might also need to balance the budget for data and compute. Spending more money on data leaves you less money for compute, and vice versa.

Data Acquisition and Annotation

The goal of data acquisition is to produce a sufficiently large dataset with the quality and diversity you need, while ensuring that your data practices respect user privacy and comply with regulations. Data acquisition involves gathering data through methods such as sourcing public data, purchasing proprietary data, annotating data, and synthesizing data. There's a niche but growing field of research in *data acquisition strategy*: how to best acquire a dataset that meets specific requirements given a budget.

The most important source of data, however, is typically data from your own application. If you can figure out a way to create a *data flywheel* that leverages data generated by your users to continually improve your product, you will gain a significant advantage.[7] Application data is ideal be-

cause it's perfectly relevant and aligned with your task. In other words, it matches the distribution of the data that you care about, which is incredibly hard to achieve with other data sources. User-generated data can be user content, system-generated data from user usage, or user feedback. How to design your user feedback system is discussed in Chapter 10.

Before investing in creating your own data, check available datasets first. Data marketplaces are vast and offer both open source and proprietary data. If you're lucky, some of them might be exactly what you need. However, it's often a mix-and-match approach. A dataset can be developed from multiple data sources via multiple acquisition channels. For example, the process of creating an (instruction, response) dataset might look as follows:

1. Find available datasets with the desirable characteristics. You might find one promising dataset with 10,000 examples.
2. Remove low-quality instructions. Let's say this leaves you with 9,000 examples.
3. Set aside the instructions with low-quality responses. Let's say you find 3,000 such examples. This leaves you with 6,000 examples of high-quality instructions and high-quality responses.
4. Manually write responses for the 3,000 high-quality instructions. Now your dataset has a total of 9,000 high-quality examples.
5. Realizing that there's not enough data for topic X, manually create a set of 100 instruction templates about X. Use an AI model to synthesize 2,000 instructions using these 10 templates.
6. Manually annotate these 2,000 synthetic instructions. Now your dataset has a total of 11,000 examples.

This is, of course, an oversimplification of the actual dataset curation process, with the vast majority of steps hidden to conserve paper and save readers from tedium. For example, there might be several steps in which you realize that many of the annotations aren't helpful, so you have to update the annotation guidelines and re-annotate your data. Worse, you might find that some of them are factually incorrect, so you have to hire another set of annotators to fact-check your original annotations. Or you might find that having 100 synthetic instructions per template hurts your data's diversity, so you have to create more templates and generate fewer instructions per template. And so on.

RESOURCES FOR PUBLICLY AVAILABLE DATASETS

Here are a few resources where you can look for publicly available datasets. While you should take advantage of available data, you should never fully trust it. Data needs to be thoroughly inspected and validated.

Always check a dataset's license before using it. Try your best to understand where the data comes from. Even if a dataset has a license that allows commercial use, it's possible that part of it comes from a source that doesn't:

1. Hugging Face and Kaggle each host hundreds of thousands of datasets.
2. Google has a wonderful and underrated Dataset Search.
3. Governments are often great providers of open data. Data.gov hosts hundreds of thousands of datasets, and data.gov.in hosts tens of thousands.
4. University of Michigan's Institute for Social Research ICPSR has data from tens of thousands of social studies.
5. UC Irvine's Machine Learning Repository and OpenML are two older dataset repositories, each hosting several thousand datasets.
6. The Open Data Network lets you search among tens of thousands of datasets.
7. Cloud service providers often host a small collection of open datasets; the most notable one is AWS's Open Data.
8. ML frameworks often have small pre-built datasets that you can load while using the framework, such as TensorFlow datasets.
9. Some evaluation harness tools host evaluation benchmark datasets that are sufficiently large for PEFT finetuning. For example, Eleuther AI's lm-evaluation-harness hosts 400+ benchmark datasets, averaging 2,000+ examples per dataset.
10. The Stanford Large Network Dataset Collection is a great repository for graph datasets.

Often, you might need to annotate your own data for finetuning. Annotation is challenging not just because of the annotation process but also due to the complexity of creating clear annotation guidelines. For example, you need to explicitly state what a good response looks like, and what makes it good. Can a response be correct but unhelpful? What's the difference between responses that deserve a score of 3 and 4? Annotation guidelines are needed for both manual and AI-powered annotations.

Some teams, including LinkedIn, have reported that annotation guidelines were among the most challenging parts of their AI engineering pipe-

line. It's alarming how often people abandon careful annotation halfway due to the time and effort required, hoping instead that their models will figure out the right responses on their own. Many models are strong enough that they can occasionally succeed, but relying on models to figure that out might be too risky for many applications.

The good news is that these guidelines are the same as those for evaluation data, as discussed in Chapter 4. This is another argument for why you should invest more time in curating evaluation guidelines and data. If you're lucky, your evaluation examples can be augmented or used as seed examples to synthesize new data. In the next section we'll discuss how to do so.

Data Augmentation and Synthesis

Together with compute and talent, data is the hardest challenge of AI. It's been a long-term goal of the whole industry to be able to generate data programmatically. Two processes commonly used are *data augmentation* and *data synthesis:*

- Data augmentation creates new data from existing data (which is real). For example, given a real image of a cat, you can flip it to create a new image of the same cat.[8]
- Data synthesis generates data to mimic the properties of real data. For example, you can simulate how a mouse moves through a web page to generate data for what bot movements would look like.

In other words, augmented data is derived from real data, whereas synthetic data isn't real. However, since the goal of both augmentation and synthesis is to automate data creation, sometimes the two terms are used interchangeably. In this chapter, I'll often use data synthesis to refer to both.

Artificially generated data has a long history in software engineering. It was originally used to generate fake data for testing purposes. For example, libraries like *Faker* and *Chance* let you generate data in simple formats such as names, addresses, phone numbers, and email addresses for testing. Let's say you've built a program to parse shipping addresses. You can use fake data generators to generate addresses in different countries and states with different formats to make sure your program can parse all of them.

With AI being capable of generating data indistinguishable from that generated by humans, it's possible to synthesize much more sophisticated data, such as doctor's notes, contracts, financial statements, product descriptions, images, video commercials, etc. This makes it easier to generate data and enables more synthetic data use cases.

While synthetic data promises to significantly reduce the pressure for human-generated data, synthetic data doesn't completely replace human data. In many use cases, as discussed in "Limitations to AI-generated data", mixing human- and AI-generated data often produces the best value.

Why Data Synthesis

Synthetic data is appealing for many reasons. You can synthesize data to improve the golden data trio: quantity, coverage, and quality. You can also synthesize data to mitigate privacy concerns and distill models:

To increase data quantity

The biggest reason for data synthesis is that it allows you to produce data at scale, promising an abundant supply of data for training and testing AI models. More data, in theory, helps models generalize to a wider range of tasks. This is especially helpful where real-world data is scarce or difficult to obtain, such as data for rare weather conditions, data for deep sea exploration, or data involving accidents for self-driving cars.

To increase data coverage

You can generate data with targeted characteristics to improve model performance or to get a model to express specific behaviors. For example, you can generate very short texts or very long texts. You can create conversations that contain toxic phrases for a toxic detection model. Vice versa, if real-world data is toxic, you can synthesize safe data. It's especially common to use AI to synthesize adversarial examples. It's also possible to generate data for the rare class to address the challenges of class imbalance. As described in "TrueTeacher", Gekhman et al. (2022) used LLMs to generate factually inconsistent summaries that they then used to train models to detect factual inconsistency.

In their paper, "Discovering Language Model Behaviors with Model-Written Evaluations" (Perez et al., 2022), Anthropic discussed various data synthesis techniques to generate specific

datasets that can test 154 different AI behaviors, including personality traits, political views, ethical stances, and social biases. They found that in head-to-head comparisons between LM (language model)-generated and human-generated datasets, "LM-written datasets approach the quality of human-written ones, sometimes even exceeding them."

In other words, you can use synthetic data to increase data coverage: generate targeted data to cover the areas where existing data is insufficient.

To increase data quality

Even though the common perception is that synthetic data is often of lower quality than human-generated data, sometimes, the reverse can be true. *Sometimes, humans might have fundamental limitations that cause human-generated data to be of lower quality than AI-generated data.* One example is tool use data discussed earlier—humans and AI have fundamentally different modes of operations and tool preferences. Another example is in generating complex math problems—AI can generate questions that are far more complex than what an average human expert might conceive.[9]

Some teams also prefer using AI to generate preference data. While each individual human can be somewhat consistent in their preference, performance across different people tends to vary significantly, influenced not only by each person's preference but also by mood and motivations. AI-generated preference ratings, in contrast, can be far more consistent and reliable.

To mitigate privacy concerns

Synthetic data is often the only option for use cases where you can't use human-generated data due to privacy concerns. For instance, in healthcare, where legislation makes it hard, if not impossible, to use real patient records to train a model, you can generate synthetic patient records that do not contain any sensitive information. In insurance, you can use synthetic claims instead of using real claims that include sensitive personal and financial information.

To distill models

Sometimes, you might want to train a model to imitate the behavior of another model. The goal is often to create a cheaper and/or faster model (the distilled model) with performance comparable to

that of the original model. This is done by training the distilled model using data generated by the original model.

These are just five of the many reasons why people turn to data synthesis. Because of its undeniable appeal, more models are being trained with synthetic data and more techniques are being developed to synthesize data.

Traditional Data Synthesis Techniques

Data synthesis isn't unique to AI. It has a long history in software testing, gaming, and robotics. Using algorithms to generate data is also called *procedural generation*, as opposed to *manual generation*. Procedural generation is commonly used in gaming to generate content such as levels, maps, items, and characters on the fly.[10] Most data generation techniques used in these industries can be applied to AI.

Traditionally, two approaches for data synthesis and augmentation have been rule-based and simulation. A newer method made possible by advanced AI models is using AI itself to synthesize data. This section gives a quick overview of these two traditional techniques before moving on to AI-powered data synthesis in the next section.

Rule-based data synthesis

The simplest way to generate data is to use predefined rules and templates. For example, to create a credit card transaction, start with a transaction template and use a random generator like Faker to populate each field in this template:

```
An example of a transaction template.
Transaction ID: [Unique Identifier]
Date: [MM/DD/YYYY]
Time: [HH:MM:SS]
Amount: [Transaction Amount]
Merchant Name: [Merchant/Store Name]
Merchant Category: [Category Code]
Location: [City, State, Country]
Payment Method: [Credit Card/Debit Card/Cash/Online Payment]
Transaction Status: [Completed/Pending/Failed]
Description: [Transaction Description]
```

Due to the sensitivity of transaction data, many fraud detection models are first trained on synthetic transaction data generated from templates

like this to prove their feasibility before being given access to real data.

It's common to use templates to generate documents that follow a specific structure, such as invoices, resumes, tax forms, bank statements, event agendas, product catalogs, contracts, configuration files, etc. Templates can also be used to generate data that follows a certain grammar and syntax, such as regular expressions and math equations. You can use templates to generate math equations for AI models to solve. DeepMind trained an Olympiad-level geometry model, AlphaGeometry, using 100 million synthetic examples (Trinh et al., 2024).

You can procedurally generate new data from existing data by applying simple transformations. For images, you can randomly rotate, crop, scale, or erase part of an image. A flipped image of a cat should still be a cat. A slightly cropped image of a soccer game should still be a soccer game. Krizhevsky et al. (2012) demonstrated in their legendary AlexNet paper the usefulness of this technique by using it to augment the ImageNet dataset (Deng et al., 2009).

For texts, you can randomly replace a word with a similar word, assuming that this replacement wouldn't change the meaning or the sentiment of the sentence. For example, the original sentence "She's a *fantastic* nurse" can generate a new example: "She's a *great* nurse".

This approach can be used to mitigate potential biases in your data. If you're concerned that there's a gender bias in your data, where, for example, the word "nurse" is associated with women while the word "doctor" is associated with men, you can replace typically gendered words with their opposites, such as "she" with "he", as shown in Table 8-2.

Table 8-2. Data augmentation can help mitigate certain biases in your data.

Original data	Augmented data
She's a fantastic nurse.	*He*'s a fantastic nurse. She's a fantastic *doctor*.
The CEO of the firm, Mr. Alex Wang, …	The CEO of the firm, *Ms. Alexa Wang*, …
Today, my mom made a casserole for dinner.	Today, my *dad* made a casserole for dinner.
Emily has always loved the violin.	*Mohammed* has always loved the violin.

Similar words can be found either with a dictionary of synonymous words or by finding words whose embeddings are close to each other in a word embedding space. You can go beyond simple word replacement by asking AI to rephrase or translate an example, as we'll discuss later.

One interesting transformation is perturbation: adding noise to existing data to generate new data. Initially, researchers discovered that perturbing a data sample slightly can trick models into misclassifying it. For example, adding white noise to a picture of a ship can cause the model to misclassify it as a car. The paper "One Pixel Attack for Fooling Deep Neural Networks" (Su et al., 2017) showed that 67.97% of the natural images in the Kaggle CIFAR-10 test dataset and 16.04% of the ImageNet test images could be misclassified by changing just one pixel. This poses a serious risk if exploited. An attacker could trick an AI model into misidentifying them as an authorized employee or make a self-driving car mistake a divider for a lane, leading to accidents.

You can train your model on perturbed data. Perturbation can both improve the model's performance and make it more robust against attacks; see Goodfellow et al., 2013 and Moosavi-Dezfooli et al., 2015). In 2019, Hendrycks and Dietterich created ImageNet-C and ImageNet-P by applying 15 common visual corruptions, such as changing brightness, adding snow, changing contrast, and adding noises to ImageNet images.

Perturbation can also be used for texts. For example, to train BERT, the authors replaced 1.5% of the tokens with random words (Devlin et al., 2018). They found this perturbation led to a small performance boost.

Visual data can be augmented using more sophisticated algorithms. Snap (2022) has a great case study on how they augment their assets to create unrepresented corner cases and mitigate implicit biases in their data. Given a character, they synthesize similar characters but with different skin colors, body types, hairstyles, clothes, and even facial expressions. These augmented assets are then used to train AI models.

Simulation

Instead of running experiments to collect data in the real world, where it can be expensive and dangerous, you can simulate these experiments virtually. For example, to test how a self-driving car reacts when encountering a horse on the highway, it'd be dangerous to release an actual horse on the highway. Instead, you simulate this situation in a virtual environment. Examples of self-driving simulation engines include CARLA

(Dosovitskiy et al., 2017), Waymo's SimulationCity, and Tesla's simulation of San Francisco.

Similarly, it's very common to simulate training data for robotics in a virtual environment. Let's say you want to train a robot to pour coffee, but you don't know exactly how each joint should move to make the action successful. You can simulate multiple scenarios with different joint movements and use only the scenarios where coffee is successfully poured to train the robot.

Simulations allow you to run multiple experiments with minimal costs while avoiding accidents and physical damage. A robot that works in simulations might not work in the real world, but if it fails in simulations, it'll likely fail in the real world. No matter how sophisticated your simulations are, however, they are simplifications of the real world. Sim2Real is a subfield that focuses on adapting algorithms that have been trained in simulations to the real world.

Simulations are common to generate data to teach models to use tools. As mentioned earlier, human-generated actions might not always be the most efficient for AI agents. Simulations might help uncover actions that humans overlook. Given a query, you can simulate different action sequences, execute these sequences, and validate their outcomes. The most efficient action sequence is then used as the annotated response for the query.

Simulations are particularly valuable for generating data for rare events. For example, in finance, researchers can simulate scenarios such as a company successfully going public or a significant bankruptcy to understand their market impacts. Manufacturers can simulate defects in materials or assemblies to generate data to train anomaly detection and quality control models. Similarly, by simulating the Earth's systems, climate scientists can create variations in temperature changes, precipitation patterns, and extreme weather scenarios. This synthetic data is then fed into AI models, enabling them to learn from a broader spectrum of possible futures.

Both rule-based and simulation-based techniques have been useful for many use cases, but it wasn't until AI become capable of generating realistic and high-quality data that data synthesis really took off. Let's look into those methods next.

AI-Powered Data Synthesis

Just as there are virtually infinite ways for humans to generate data, AI can also do so in many ways. The techniques discussed here are not comprehensive, but they should give you a good overview.

Powerful AI models open many new possibilities for simulations. AI can simulate the outcomes of arbitrary programs. For example, "StableToolBench" (Guo et al., 2024) demonstrates how to use AI to simulate APIs without having to evoke them. Imagine you want to train a model to interact with a set of APIs. Instead of making actual API calls—which might be costly or slow—you can use an AI model to simulate the expected outcomes of those calls.

AI can simulate humans. For example, imagine you want to train a bot to play chess. A game played by humans might take too long. Matches with AI players would be much faster. To train its Dota 2 bot, OpenAI used a simulator that enabled the bot to play approximately 180 years' worth of games every day. The bot learned by playing against itself, an approach called *self-play*, which helped it develop and refine strategies over time (OpenAI, 2019). Similarly, DeepMind used self-play to collect data from millions of Go games to train AlphaGo (Silver et al., 2016).

Self-play is useful not just for game bots but also for general agents. You can have AIs negotiate against each other using different strategies to see which one works better. You can have one version of the model play the role of a customer with issues and another play the customer support agent.

AI's paraphrasing and translation abilities can be used to augment existing datasets. For example, given the query "How to reset my password?", AI can paraphrase it to create three new queries:

1. "I forgot my password."
2. "How can I change my password?"
3. "Steps to reset passwords."

Yu et al. (2023) rewrote the 15,000 examples in MATH and GSM-8K in different ways to create MetaMath, a new dataset of almost 400,000 examples. They showed that their models, trained on this new dataset, outperformed larger models on related math benchmarks.

It's common to use AI to translate data in high-resource languages (more available online) into low-resource languages to help train models in low-

resource languages. This is useful for training a small model specializing in a low-resource language like Quechua or Lao.

You can verify the quality of translations with *back-translation*. Let's say the original English sentence is X and the translated Lao sentence is Y. You can use another model to translate the translation back into the original language, X', then compare X' with the original sentence X. If they are very different, the translation Y is likely bad.

AI can translate not just natural languages but also programming languages. You can use AI to translate code written in one language to another. The Llama 3 authors used code translation of their SFT dataset with a wider range of programming languages. In fact, the training of Llama 3 depends heavily on synthetic data, and the authors used many creative techniques to generate useful data.

For example, they used back-translation to generate code explanations and documentation. Starting with code snippets, they used AI to generate explanations and documentation. They then again used AI to generate code snippets from the explanations and documentation. Only if the generated code is considered faithful to the original will the explanation and documentation be used to finetune the model.

AI can generate data for both pre-training and post-training, though synthetic data is intentionally included much more often in post-training than in pre-training. One possible explanation for this is that pre-training's goal is to increase the model's knowledge, and while AI can synthesize existing knowledge in different formats, it's harder to synthesize new knowledge.

However, as the internet becomes flooded with AI-generated content, models that rely on internet data are likely already pre-trained on synthetic data. There are also synthetic datasets such as Cosmopedia (Allal et al., 2024), a 25-billion-token collection of synthetic textbooks, blog posts, stories, posts, and WikiHow articles generated by Mixtral-8x7B-Instruct-v0.1 (Jiang et al., 2024).

Data synthesis for post-training is also more common because post-training data, including both instruction data and preference data, generally demands the most effort to produce. Using AI to pick the better response among several responses is more straightforward—much of it was already covered in Chapter 3. The main challenge is to take into account the model's biases, such as first-position bias, where the model is more likely to prefer the first option. To avoid this, NVIDIA researchers asked the AI

judge twice, once with the response order swapped. They picked a valid (prompt, winning, losing) triplet only when the AI judge picked the same winner both times (NVIDIA, 2024).

The next section will focus on how to use AI to synthesize instruction data for supervised finetuning.

Instruction data synthesis

During instruction finetuning, each example includes an instruction and a response. AI can be used to synthesize the instructions, the responses, or both. For example, you can use AI to generate instructions and humans to write responses. You can also use humans to write instructions and AI to generate responses:

- For instruction generation, to ensure that you generate sufficient instructions to cover your use case, you can start with a list of topics, keywords, and/or the instruction types you want in your dataset. Then, for each item on this list, generate a certain number of instructions. You can also begin with a set of templates and generate a certain number of examples per template. Note that both the topic list and templates can be generated by AI.
- For response generation, you can generate one or more responses per instruction.

For instance, to create UltraChat (Ding et al., 2023), a multi-turn dialogue dataset, the authors first asked ChatGPT to generate 30 topics about various aspects of our daily lives, such as technology, food and drink, fashion, nature, education, finance, travel, etc. For each topic, they asked ChatGPT to generate 30 to 50 subtopics. The authors then used the same model to generate instructions and corresponding responses for these subtopics.

Similarly, to train Alpaca (Taori et al., 2023), Stanford researchers began with 175 (instruction, response) examples from the Self-Instruct seed dataset (Wang et al., 2022). These examples were originally written to cover a diverse and interesting range of uses. Alpaca authors then used a GPT-3 model, *text-davinci-003*, to generate 52,000 (instruction, response) pairs that mirrored these seed examples, as shown in Figure 8-5.

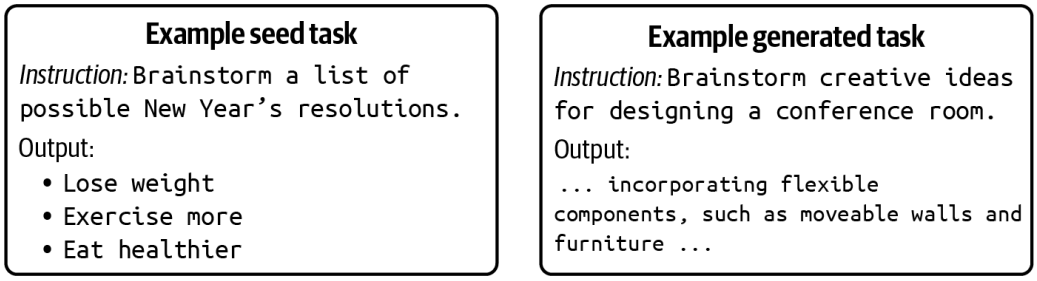


Figure 8-5. A seed task and a generated task used to train Alpaca.

There are also many creative ways to synthesize instruction data with certain characteristics. For example, just like it's harder for humans to write longer content than shorter content, it's harder for AI to generate high-quality long responses than short instructions. The longer the response, the more chance AI has to hallucinate. What if we use human-generated responses with AI-generated instructions? Some researchers, such as Köksal et al. (2023), Li et al. (2023), and Chen et al. (2023), follow the *reverse instruction* approach: take existing long-form, high-quality content like stories, books, and Wikipedia articles and use AI to generate prompts that would elicit such content. This yields higher-quality instruction data, avoiding AI-generated hallucinations in the responses.

It's possible to use reverse instruction to develop increasingly powerful models without adding manually annotated data.[11] Li et al. (2023) shows how this works:

1. Start with a small number of seed examples to train a weak model.
2. Use this weak model to generate instructions for existing high-quality content to create high-quality instruction data.
3. Finetune the weak model with this new high-quality instruction data.
4. Repeat until desirable performance is reached.

A creative approach is to use synthetic data to finetune a model for understanding longer contexts. For example, if your current model processes a maximum of 8K tokens but you want it to handle 128K tokens, the long-context finetuning process might look like this:

- Split long documents into shorter chunks (e.g., under 8K tokens).
- For each short chunk, generate several (question, answer) pairs.
- For each (question, answer) pair, use the original long document, which may exceed 8K tokens but be shorter than your target length, as the context. This trains the model to use the extended context to answer questions.

The level of detail in the Llama 3 paper (Dubey et al., 2024) makes it an excellent case study for instruction data synthesis. I've already men-

tioned two ways in which Llama 3 synthesized data: code translation and code back-translation. Both of these methods generate more data from existing code snippets. However, the authors also used AI to synthesize coding instruction data from scratch, using the following workflow:

1. Use AI to generate a large collection of programming problem descriptions that span a diverse range of topics.
2. Given a problem description and a programming language, generate a solution. Dubey et al. found that including general rules of good programming and CoT reasoning helped improve response quality.

To ensure the quality of the generated data, they employed a rigorous correctness analysis and error correction pipeline:

1. Run generated code through parsers and linters to catch syntactic errors such as missing imports and uninitialized variables.
2. Use unit tests to catch runtime execution errors. Interestingly enough, they used AI to generate these unit tests.
3. When a solution fails at any step, prompt the model to revise the code. The prompt included the original problem description, the faulty solution, and feedback from the parser, linter, and unit tests. Only examples that pass all checks are included in the final supervised finetuning dataset.[12]

Combining all three methods together—code translation, code back-translation, and code generation—Llama 3's data synthesis workflow is quite impressive. To summarize, here's how these three methods work together:

1. Use AI to generate problem descriptions.
2. Use AI to generate solutions for each problem in different programming languages.
3. Use AI to generate unit tests to test the generated code.
4. Prompt AI to fix errors in the synthesized code.
5. Use AI to translate generated code to different programming languages. Filter out translated code that doesn't pass tests.
6. Use AI to generate conversations about the code, including code explanation and adding documentation. Filter out generated explanations and documentation that doesn't pass back-translation verification.

Using this pipeline, Dubey et al. were able to generate over 2.7 million synthetic coding-related examples for the supervised finetuning of Llama 3.1.

Data verification

Given the importance of data quality in the model's performance, it's crucial that we have a way to verify the quality of data. The quality of AI-generated data can be measured the same way you'd evaluate other AI outputs—by functional correctness and AI judges.

While this section focuses on synthetic data, most of the techniques can be used to evaluate the quality of training data in general.

Recall the concept of evaluation-driven development from Chapter 4, where companies are more likely to create applications they can evaluate. Similarly, people tend to synthesize data they can verify. Coding is one of the most popular foundation model use cases because it can be functionally evaluated, and for the same reason, coding-related examples are among the most commonly synthesized data. Most of the synthetic data used to train Llama 3 is coding-related. All three methods the authors used to synthesize data result in data that can be programmatically verified, x, by code execution and back-translation.

For synthetic data that can't be verified by functional correctness, it's common to use AI verifiers. An AI verifier can be a general-purpose AI judge or a specialized scorer. There are many ways to frame the verification problem. In the simplest form, the AI verifier can assign each generated example a score from 1 to 5 or classify each example as good or bad. You can also describe to a foundation model the quality requirements and instruct the model to determine if a data example meets these requirements.

If you care about the factual consistency of data, you can use the factual inconsistency detection techniques discussed in Chapter 4 to filter out examples that are likely to contain hallucinations.

Depending on the use case and the generated data, you can also get creative. For instance, if you want synthetic data to mimic real data, its quality can be measured by how difficult it is to distinguish between the two. You could train an AI content detector to identify AI-generated data—if it's easy to differentiate between real and synthetic data, the synthetic data isn't good. Or, if you want the synthetic data to resemble high-quality academic work, you could train a classifier to predict whether a generated paper would be accepted at a prestigious conference like NeurIPS (the Conference and Workshop on Neural Information Processing Systems) and discard any papers predicted to be clear rejects.

You can have a model to detect the topic of each generated example and then remove examples whose topics are irrelevant to your task. If you expect all data to follow a similar pattern, you can also use anomaly detection to identify outliers—outlier examples might be of low quality.

Just like real data, synthetic data can also be filtered using heuristics. In general, you might want to remove examples that are empty or too short for your application. If an example is too long, you might want to truncate or remove it. You can filter out data by keywords, by user/author, by creation date, by metadata, or by source. For example, the Self-Instruct authors (Wang et al., 2022) filtered out generated examples using the following heuristics:

- Repetitive examples
- Instructions that are too long or too short
- Examples with the same instruction but different responses
- Examples where the output is a repetition of the input

Even though there are many techniques to evaluate synthetic data, evaluation remains challenging. As with other AI applications, the ultimate quality test for AI-generated data is its real-world performance—whether it can improve the model's performance—and synthetic data has passed this test for many models.

Limitations to AI-generated data

Given the increasing usefulness of synthetic data, it's exciting to imagine the possibility of never having to worry about human-annotated data again. However, while the role of synthetic data will certainly continue to grow in importance over time, AI-generated data might never entirely replace human-generated data. There are many reasons why, but the four major ones are the difference in quality, the limitations of imitation, potential model collapse, and the way AI generation of data obscures its lineage.

Quality control

AI's generated data can be of low quality, and, as people never tire of saying, "garbage in, garbage out." As mentioned earlier, people will be hesitant to use synthetic data if they can't verify its quality. Being able to develop reliable methods and metrics to evaluate data will be essential in making synthetic data more useful.

Superficial imitation

As warned by "The False Promise of Imitating Proprietary LLMs" (Gudibande et al., 2023), the perceived performance achieved by mimicking might be superficial. This research shows that the imitation models are good at mimicking the style of the teacher models but might struggle with factual accuracy and generalization to tasks outside the training data.

Worse, imitation can force the student model to hallucinate. Imagine if the teacher model is capable of answering complex math questions, so its responses to those questions are solutions. Training a student model on these solutions effectively teaches it to produce answers that look like solutions, even if the student model isn't capable of solving these questions.[13] Gudibande et al. (2023) suggest that for improvement in reasoning capabilities, we need to focus on improving the quality of the base models.

Potential model collapse

It's also unclear how much AI-generated data a model can train on. Some studies have shown that *recursively* using AI-generated data in training causes irreversible defects in the resulting models, degrading their performance over time. In "The Curse of Recursion: Training on Generated Data Makes Models Forget", Shumailov et al. (2023) named this phenomenon *model collapse* and demonstrated its occurrences in models including Variational Autoencoders, Gaussian mixture models, and LLMs. Model collapse can happen during both pre-training and post-training.[14]

One possible explanation is that AI models are more likely to generate probable events (e.g., not having cancer) and less likely to generate improbable events (e.g., having cancer). Over multiple iterations, probable events become over-represented, whereas improbable events become under-represented in the generated data. This causes models to output more common events over time while forgetting rare events.

In "Is Model Collapse Inevitable?" Gerstgrasser et al. (2024) argue that while model collapse is inevitable if the entire training dataset is synthetic, it can be avoided by mixing synthetic data with real data. Bertrand et al. (2023) and Dohmatob et al. (2024) show similar results. However, none of these papers has a definitive recommendation for the proportion of synthetic data to real data.

Some people have been able to improve model performance using a large amount of synthetic data. For example, "Common 7B Language Models Already Possess Strong Math Capabilities" (Li et al., 2024) demonstrates that synthetic data is nearly as effective as real data in finetuning Llama 2-7B models on math problems. In their experiments, synthetic data shows no clear saturation when scaled up to approximately one million samples. Similarly, Nemotron-4 340B-Instruct (NVIDIA, 2024) used 98% synthetic data during its instruction finetuning and preference finetuning phase. However, these experiments were carried out for only one model iteration.

AI-generated data might also perpetuate biases. "Data Feedback Loops: Model-driven Amplification of Dataset Biases" (Taori and Hashimoto, 2023) demonstrates that when models are trained on datasets that include previous model outputs, any existing biases in the model can be amplified. The authors find that the more faithful the model's outputs to the characteristics of the original training distribution, the more stable the feedback loop, thus minimizing the risk of bias amplification.

Obscure data lineage

This limitation of AI-generated data is more subtle. AI generation obscures data lineage. AI models are influenced by their training data and can sometimes regurgitate it without the user knowing. This creates risks. Let's say you use model X to generate data to train your model. If model X was trained on data with copyright violations, your model might also violate copyrights.

Or imagine you then use benchmark B to evaluate your model, which shows a strong performance. However, if model X was also trained on benchmark B, your result on B is contaminated. Without clear data lineage, it's hard to assess a model's commercial viability or trust its performance.

We've discussed how to use AI to generate data and how to evaluate the generated data, as well as its limitations. In the next section, let's switch gears to discuss one special use case of data synthesis where AI-generated data isn't just supplementary but is required: model distillation.

Model Distillation

Model distillation (also called *knowledge distillation*) is a method in which a small model (student) is trained to mimic a larger model (teacher)

(Hinton et al., 2015). The knowledge of the big model is distilled into the small model, hence the term distillation.

Traditionally, the goal of model distillation is to produce smaller models for deployment. Deploying a big model can be resource-intensive. Distillation can produce a smaller, faster student model that retains performance comparable to the teacher. For example, DistilBERT, a model distilled from BERT, reduces the size of a BERT model by 40% while retaining 97% of its language comprehension capabilities and being 60% faster (Sanh et al., 2019).

The student model can be trained from scratch like DistilBERT or finetuned from a pre-trained model like Alpaca. In 2023, Taori et al. finetuned Llama-7B, the 7-billion-parameter version of Llama, on examples generated by *text-davinci-003*, a 175-billion-parameter model. The resulting model, Alpaca, behaves similarly to *text-davinci-003*, while being 4% the size of the teacher model.

NOTE

Not all models can be distilled. Many model licenses prohibit using their outputs to train other models, particularly to train competing models.

Synthetic instruction data is commonly used together with adapter-based techniques, such as LoRA. For example, BuzzFeed finetuned a Flan-T5 model using LoRA and examples generated by OpenAI's *text-davinci-003*. The resulting model reduced their inference cost by 80%, though it was unclear how well the model performed (2023).

Note that not all training with synthetic data is model distillation. Model distillation implies that the teacher model's performance is the student's gold standard. However, it's possible to use synthetic data to train a student model that is larger and more powerful than the teacher.

Model bootstrapping with reverse instruction (Li et al., 2023), discussed in the previous section, is one example. Another example is NVIDIA's Nemotron-4. A team of NVIDIA researchers first pre-trained a 340B parameter base model. This base model was then finetuned using instruction and preference data generated by Mixtral-8x7B-Instruct-v0.1 (Jiang et al., 2024), a 56-billion-parameter mixture-of-experts model.[15] The resulting student model, Nemotron-4-340B-Instruct, outperformed the teacher model on a variety of tasks (NVIDIA, 2024).

The Llama 3 paper notes that while training on data generated by a more competent model can significantly improve a model's performance, training indiscriminately on self-generated data doesn't improve the model's performance and can even degrade it. However, by introducing mechanisms to verify the quality of synthetic data and using only verified synthetic data, they were able to continually improve a model using its generated data.

Data Processing

Data needs to be processed according to the requirements of each use case. This section discusses some data processing steps for reference.

I find it helpful to read model papers that disclose their dataset details, as they often contain great tips on how the researchers curated, generated, and processed data.

TIP

With a large amount of data, each of these processing steps can take hours, if not days. Tips to help optimize efficiency during the process include:

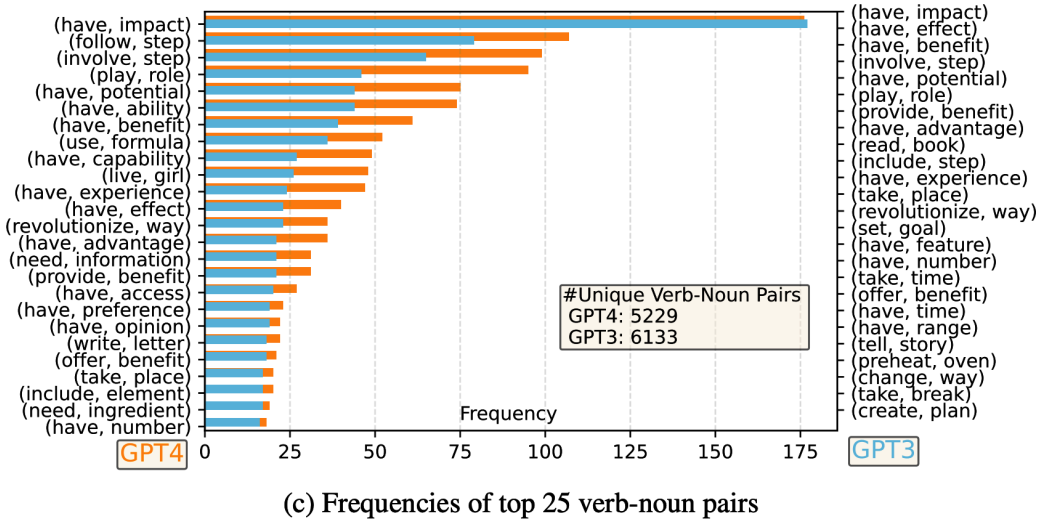
- You can do these data processing steps in whichever order saves time and compute. For example, if it takes more time to clean each example than to deduplicate data, you might want to remove the duplicated examples first before cleaning them. But if deduplication takes more time than filtering out low-quality data, filter out low-quality data first.
- Always do trial runs to validate that your processing scripts work as expected before applying the scripts to all your data.
- Avoid changing data in place. Consider keeping a copy of the original data for two reasons:
  - You or another team might need to process the data in different ways for other applications.
  - Bugs in your scripts can potentially corrupt your data.

Inspect Data

Let's say that after combing through public and internal data, you've gathered a raw dataset. The first thing to do is inspect the data to get a sense of its quality. Get the data's information and statistics. Where does the data come from? How has it been processed? What else has it been used for?

Plot the distribution of tokens (to see what tokens are common), input lengths, response lengths, etc. Does the data use any special tokens? Can you get a distribution of the topics and languages in the data? How relevant are these topics and languages to your task?

You can be creative in the statistics to use to understand your data. For example, a group of Microsoft researchers (2023) used the distribution of (verb, direct object, noun) pairs and response length to compare the difference between GPT-3's and GPT-4's generations for the same set of instructions, as shown in Figure 8-6 and Figure 8-7. This type of analysis is helpful not only to evaluate data but also to evaluate models.



(c) Frequencies of top 25 verb-noun pairs

Figure 8-6. One statistic you can use is the distribution of (verb, direct object noun) in your data. Image from "Instruction Tuning with GPT-4" (Peng et al., 2023).

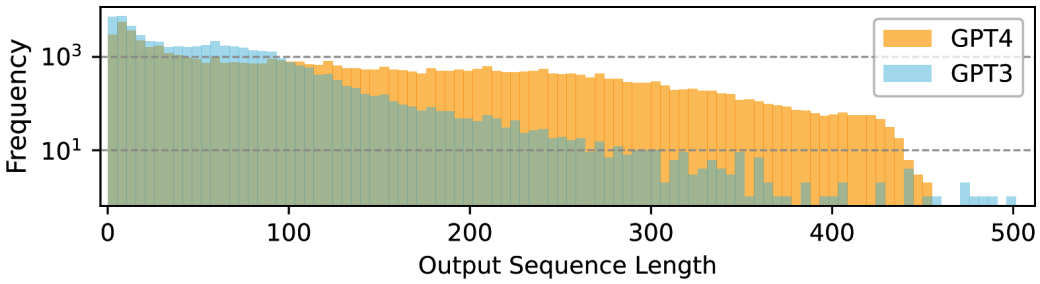


Figure 8-7. The distribution of response length for GPT-4 and GPT-3. Image from "Instruction Tuning with GPT-4" (Peng et al., 2023).

GPT-4 seems to have a broader and more diverse range of verb-noun pairings and tends to generate longer responses.

Plot these distributions by data source, time, annotator, etc. Do you notice any question patterns that tend to get longer/shorter responses or higher/lower scores? Are there any outliers? What might be the cause of these outliers? What to do with them?

If the scores are supposed to follow a normal distribution, do scores by all annotators follow a normal distribution? You might notice that some an-

notators tend to give much shorter responses or bias toward higher scores, and it's up to you to decide what to do with their annotations.

If each example has more than one annotation, compute the inter-annotator disagreement. Check the examples with conflicting annotations and resolve the conflicts.

There are many data exploration tools you should use, but they won't be replacements for manual data inspection. In every project I've worked on, *staring at data for just 15 minutes usually gives me some insight that could save me hours of headaches*. Greg Brockman, an OpenAI co-founder, tweeted: "Manual inspection of data has probably the highest value-to-prestige ratio of any activity in machine learning."

Look at your data to see if the examples make sense. If it's annotated data, pick out a few queries and try to annotate them yourself to see if your annotations match the given annotations. This will give you a sense of how trustworthy the annotations are. Fact-check the responses. How unique are the examples? Are there any examples with the same query but with different responses? Are there any examples with the same responses but with different queries?

Deduplicate Data

Duplicated data can skew the data distribution and introduce biases into your model. Imagine a dataset that looks like Table 8-3. The duplicated entries might lead the model to the wrong conclusion that all red-colored items should be expensive. Duplications can cause test set contamination. When splitting duplicated data into train and test sets, one example might be in the train set and its duplicate in the test set.

Table 8-3. A toy dataset with duplicate examples in grey cells.

	Input (Product description)	Output (Price)
1	`{item: pencil, color: red}`	`$20`
2	`{item: compass, color: green}`	`$2`
3	`{item: pencil, color: red}`	`$20`
4	`{item: pencil, color: red}`	`$20`
5	`{item: pencil, color: green}`	`$1`

Multiple studies have shown the negative impact of training data duplications on model performance; see Lee et al. (2021) and Tirumala et al.

(2023). An Anthropic study demonstrated that repeating 0.1% of the data 100 times can cause an 800M parameter model's performance to degrade to that of a 400M parameter model despite the other 90% of the training tokens remaining unique (Hernandez et al., 2022). Even when duplications don't hurt your model's performance, they can waste your time and compute.

Depending on the data, there are many forms of duplication, some of which are harder to detect. For example, here are a few types of duplications in a dataset of documents:

- Whole document duplications: the same document appearing more than once.
- Intra-document duplications: e.g., the same paragraph appears twice in one document.
- Cross-document duplications: e.g., the same popular quote appears in multiple documents.

What can be considered duplications also depends on your definition. For example, do you want to deal with duplications at the document level, paragraph level, sentence level, or token level? Would two texts have to match exactly to be considered duplicates, or would an 80% overlap be sufficient? Are two lists considered duplicates if they have the same items but in different order?

The task of deduplication can leverage the same techniques used for similarity measurements (discussed in Chapter 3). Data deduplication is also used for identity resolution, determining whether two identities (e.g., two social media profiles) are the same. Here are some concrete ways you can deduplicate data:

Pairwise comparison

Compute the similarity score of each example to every other example in the dataset, using exact match, n-gram match, fuzzy match, or semantic similarity score, as discussed in Chapter 3. This approach can be expensive with large datasets, however.

Hashing

Hash examples into different buckets and check only among examples that fall into the same bucket. Hash-related deduplication methods include MinHash and Bloom filter.

Dimensionality reduction

Use a dimensionality reduction technique to first reduce the dimensions of your data and then do a pairwise comparison. Many techniques used for vector search, as discussed in Chapter 6, can be used for this.

A quick search will return many libraries that help with deduplication. Some of them are dupeGuru, Dedupe, datasketch, TextDistance, TheFuzz, and deduplicate-text-datasets.[16]

Clean and Filter Data

Data needs to be cleaned to make your model performant and safe.

First, you might want to remove extraneous formatting tokens. Since many public datasets are scraped from the internet, extraneous HTML tags are quite common. Unless you want to train your model on HMTL tags, remove them. Databricks found that removing extraneous Markdown and HTML tokens improved their model's accuracy by 20% while reducing their input token lengths by 60%.

You need to clean your data of anything that isn't compliant with your policies, such as PII, sensitive data, copyrighted data, or data that is considered toxic. Techniques discussed in Chapter 4 can help. Remove all the fields that you're not allowed to use, such as zip code, name, and gender.

You also might want to remove low-quality data, using techniques discussed in "Data verification" to detect low-quality data.

Manual inspection of data is especially important in this step. Staring at data might help you notice patterns that you can use as heuristics to detect low-quality data. Heuristics to detect low-quality data might be non-obvious. For example, Kern et al. (2024) found that annotations made in the second half of an annotation session are of lower quality, likely due to annotator boredom or fatigue.

If there is more data than you need or can afford to use (e.g., due to your compute budget), you can further filter your data. For example, you can use *active learning* techniques to select examples that are the most helpful for your model to learn from. You can also use importance sampling to find examples that are most important to your task. Their efficiencies depend on whether you have a good way to evaluate the importance of each training example. Meta researchers, in their paper on data pruning (Sorscher et al., 2022), concluded that the discovery of good data-pruning

metrics can significantly reduce the resource costs of modern deep learning.

Format Data

Once you've deduplicated and cleaned your data, you need to get it into the right format expected by the model you're finetuning. Each model uses a specific tokenizer and expects data in a specific chat template, as discussed in Chapter 5. Getting data into the wrong chat template can cause strange bugs in your model.

If you're doing supervised finetuning, your data is most likely in the format (instruction, response). Instructions can be further decomposed into (system prompt, user prompt). If you've graduated to finetuning from prompt engineering, the instructions used for finetuning might be different from the instructions used during prompt engineering. During finetuning, instructions typically don't need task descriptions or examples. If you have sufficient training examples, the model can learn the expected behavior of the task from the examples directly.

As an example, imagine that you've been using this three-shot instruction for your food classification task with a base model:

```
Label the following item as either edible or inedible.

Item: burger
Label: edible

Item: car
Label: inedible

Item: mushroom
Label: edible

Item: {INPUT}
Label:
```

For finetuning, all the examples included in the 3-shot prompt can be converted into training examples. The training data for finetuning will look like Table 8-4.

Table 8-4. Example training data used for a food classification task.

Example ID	Input	Output
1	`burger -->`	`edible`
2	`car -->`	`inedible`
3	`mushroom -->`	`edible`
...	...	...

Once the model is finetuned, you can use a prompt as simple as:

```
{INPUT} -->
```

This is much shorter than the prompt used with the base model. Therefore, if you're worried about the input tokens of your instructions, finetuning can be one way to help manage the cost.

Different finetuning data formats can impact your finetuned model's performance. Experiments to determine the best format for you can be helpful.

When you use the finetuned model, make sure that the prompts you use match the format of the finetuning data. For example, if the training data uses the prompt in the format "burger -->", any of the following prompts can cause issues:

- "burger": missing the end arrow
- "Item: burger -->": extra prefix
- "burger --> ": extra space appended

Summary

Even though the actual process of creating training data is incredibly intricate, the principles of creating a dataset are surprisingly straightforward. To build a dataset to train a model, you start by thinking through the behaviors you want your model to learn and then design a dataset to show these behaviors. Due to the importance of data, teams are introducing dedicated data roles responsible for acquiring appropriate datasets while ensuring privacy and compliance.

What data you need depends not only on your use case but also on the training phase. Pre-training requires different data from instruction finetuning and preferred finetuning. However, dataset design across training phases shares the same three core criteria: quality, coverage, and quantity.

While how much data a model is trained on grabs headlines, having high-quality data with sufficient coverage is just as important. A small amount of high-quality data can outperform a large amount of noisy data. Similarly, many teams have found that increasing the diversity of their datasets is key to improving their models' performance.

Due to the challenge of acquiring high-quality data, many teams have turned to synthetic data. While generating data programmatically has long been a goal, it wasn't until AI could create realistic, complex data that synthetic data became a practical solution for many more use cases. This chapter discussed different techniques for data synthesis with a deep dive into synthesizing instruction data for finetuning.

Just like real data, synthetic data must be evaluated to ensure its quality before being used to train models. Evaluating AI-generated data is just as tricky as evaluating other AI outputs, and people are more likely to use generated data that they can reliably evaluate.

Data is challenging because many steps in dataset creation aren't easily automatable. It's hard to annotate data, but it's even harder to create annotation guidelines. It's hard to automate data generation, but it's even harder to automate verifying it. While data synthesis helps generate more data, you can't automate thinking through what data you want. You can't easily automate annotation guidelines. You can't automate paying attention to details.

However, challenging problems lead to creative solutions. One thing that stood out to me when doing research for this chapter is how much creativity is involved in dataset design. There are so many ways people construct and evaluate data. I hope that the range of data synthesis and verification techniques discussed in this chapter will give you inspiration for how to design your dataset.

Let's say that you've curated a wonderful dataset that allows you to train an amazing model. How should you serve this model? The next chapter will discuss how to optimize inference for latency and cost.

[1] The increasing importance of data is reflected in how data effort changed from GPT-3 to GPT-4. In the contribution list for GPT-3 (OpenAI, 2020), only two people were credited with data collecting, filtering, and deduplicating, and conducting overlap analysis on the training data. This dramatically changed three years later. For GPT-4 (OpenAI, 2023), eighty people were credited for being involved in different data processes. This list doesn't yet include data annotators that OpenAI contracted through data providers. For something that sounds as simple as a ChatML format, eleven people were involved, and many of them are senior researchers. Back in their 2016 AMA (ask me anything) thread, Wojciech Zaremba, one of OpenAI's cofounders, said that they intended to conduct most of their research using publicly available datasets.

[2] If you use a lot of data, ensuring data compliance alone can be a full-time job.

[3] While I love writing, one of the things I absolutely do not enjoy is trying to condense everyone's opinions into one single definition. IBM defined data quality along seven dimensions: completeness, uniqueness, validity, timeliness, accuracy, consistency, and fitness for purpose. Wikipedia added accessibility, comparability, credibility, flexibility, and plausibility. Many of these definitions focus on data quality in a broad range of use cases. Here, I want to focus on data quality for finetuning.

[4] One painful bug I still remember is when a float column in my data was wrongly stored as integers, which round these values, leading to perplexing behaviors.

[5] While this doesn't refer to the uniqueness of your data, having data that nobody else has can be extremely valuable.

[6] In *Designing Machine Learning Systems*, I also covered other techniques to reduce the demand for annotated data, including weak supervision, semi-supervision, and active learning.

[7] I've heard so many companies talking about data flywheels in their pitches that I'm convinced it isn't legal to start an AI startup without mentioning the data flywheel.

[8] My book, *Designing Machine Learning Systems*, discusses data augmentation in Chapter 4.

[9] One obvious example that I didn't include in the main text is when you want to train a model to detect AI-generated content. You need AI-generated content as training examples.

[10] Many awesome games are possible only because of procedural generation. Games like *Minecraft* and *No Man's Sky* use noise functions and fractal algorithms to create vast, immersive worlds. In *Dungeons & Dragons*, procedural generation can be used to create random dungeons, quests, and encounters, making the game more appealing by adding an element of unpredictability and endless possibilities.

[11] The implication of this is that, in theory, it's possible to train a model that can continually improve upon itself. However, whether this is possible in practice is another story.

[12] They "observed that about 20% of solutions were initially incorrect but self-corrected, indicating that the model learned from the execution feedback and improved its performance."

[13] The same issue can happen with human annotations. If the human labeler uses the knowledge they have but the model doesn't to answer a question, they are effectively teaching the model to hallucinate.

[14] The concept was also later explained by the same authors in "AI Models Collapse When Trained on Recursively Generated Data" (*Nature*, July 2024).

[15] Comparing the parameter count of a mixture-of-experts model like Mixtral to that of a dense model like Nemotron-4 isn't fair, but the point that the teacher model (Mixtral) is smaller than the student model (Nemotron-4) still holds.

[16] One of my open source libraries, lazyNLP, also supports overlap estimation and deduplication using Bloom filter.