

Data Collection + Evaluation

Sourcing and evaluating the data used to train AI involve important considerations. This chapter covers the following questions:

Does our training dataset have the features and breadth to ensure our AI meets our users' needs?

Should we use an existing training dataset or develop our own?

How can we ensure that the data quality is high?

How can we work with labelers to prevent errors and bias in datasets when generating labels?

Are we treating data workers fairly?

Want to drive discussions, speed iteration, and avoid pitfalls? [Use the worksheet.](#)

What's new when working with AI

In order to make predictions, AI-driven products must teach their underlying machine learning model to recognize patterns and correlations in data. This data is called training data, and can be collections of images, videos, text, audio and more. You can use existing data sources or collect new data expressly to train your system. For example, you might use a database of responsibly crowdsourced data on plants from different regions around the world to train an AI-powered app that recognizes plants that are safe to touch.

The training data you source or collect, and how those data are labeled, directly determines the output of your system – and the quality of the user experience. Once you're sure that using AI is indeed the right path for your product (see User Needs + Defining Success) consider the following:

- ① **Plan to gather high-quality data from the start.** Data is critical to AI, but more time and resources are often invested in model development than data quality. You'll need to plan ahead as you gather and prepare data, to avoid the effects of poor data choices further downstream in the AI development cycle.
- ② **Translate user needs into data needs.** Determine the type of data needed to train your model. You'll need to consider predictive power, relevance, fairness, privacy, and security.

- ③ **Source your data responsibly.** Whether using pre-labeled data or collecting your own, it's critical to evaluate your data and their collection method to ensure they're appropriate for your project.
- ④ **Prepare and document your data.** Prepare your dataset for AI, and document its contents and the decisions that you made while gathering and processing the data.
- ⑤ **Design for labelers & labeling.** For supervised learning, having accurate data labels is crucial to getting useful output from your model. Thoughtful design of labeler instructions and UI flows will help yield better quality labels and therefore better output.
- ⑥ **Tune your model.** Once your model is running, interpret the AI output to ensure it's aligned with product goals and user needs. If it's not, then troubleshoot: explore potential issues with your data.

① Plan to gather high-quality data from the start

Data is critical to AI, but the time and resources invested in model development and performance often outweigh those spent on data quality.

This can lead to significant downstream effects throughout the development pipeline, which we call "data cascades". Here's a hypothetical example of a data cascade:

Let's say you're developing an app that allows the user to upload the picture of a plant, and then displays a prediction for the plant's type, and whether it is safe for humans and pets to touch and eat it.

When you prepared a dataset to train the image classification model, you used mostly images of plants native to North America because you found a dataset that was already labeled and easy to use to train the model.

Once you released the Plant Pal app, however, you found out that many users were reporting plant detection errors in South America.

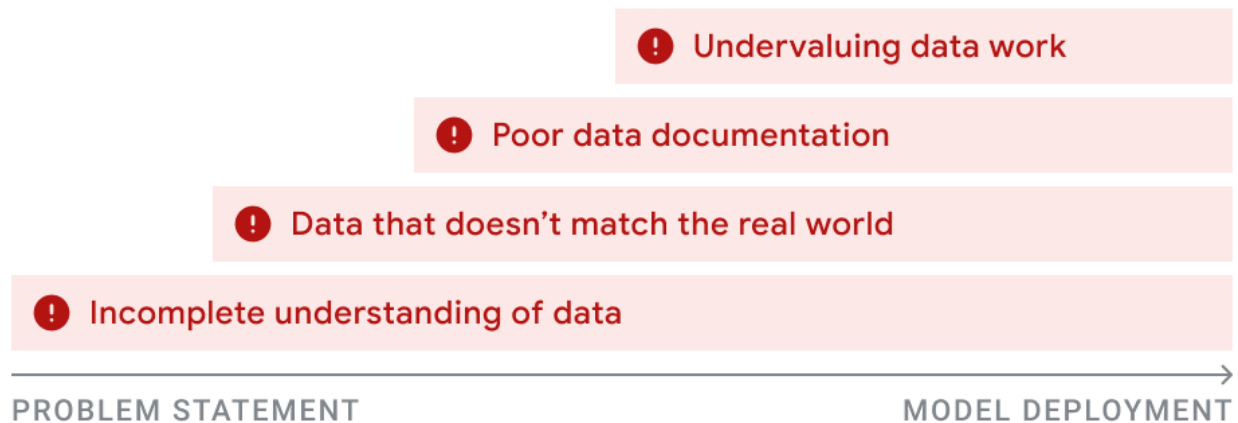
This is an example of a data cascade, as the effects of the mismatch between training data and user data in the real world were delayed. This cascade could have been avoided by including images of plants native to South America when developing the AI model, or releasing the app to users in North America only.

Some data cascades may be hard to diagnose, and you may not see their impact until users report a poor experience.

It's best to plan to use high-quality data from the beginning, whether you're creating a dataset from scratch or reusing existing datasets, and good planning and interrogating your dataset can help you detect issues earlier. High-quality data can be defined as:

- Accurately representing a real-world phenomenon or entity
- Collected, stored, and used responsibly
- Reproducible

- Maintainable over time
- Reusable across relevant applications
- Having empirical and explanatory power



Example of how data issues can compound into data cascades

Most roles on a product team and their data partners are involved in the data pipeline when developing with AI. Understanding their priorities, workflows, and challenges can point us to solutions for higher-quality data.

② Translate user needs into data needs

Datasets that can be used to train AI models contain examples, which contain one or more features, and possibly labels.

		FEATURES				
		Runner ID	Run	Runner time	Elevation	Fun
EXAMPLES	AV3DE	Boston Marathon	03:40:32	1,300ft	Low	LABELS
	X8KGF	Seattle Oktoberfest 5k	00:35:40	0ft	High	
	BH9IU	Houston half-marathon	02:01:18	200ft	Medium	

The scope of features, the quality of the labels, and representativeness of the examples in your training dataset are all factors that affect the quality of your AI system.

The table above contains data about races that an app could use to train an ML model to predict how enjoyable a given race will be. Here's how examples, features and labels could affect the quality of that model:

Examples

If examples used to train the run recommendation algorithm only come from elite runners, then they would likely not be useful in creating an effective model to make predictions for a wider user base. However, they may be useful in creating a model geared towards elite runners.

Features

If the elevation gain feature was missing from the dataset, then the ML model would treat a 3.0 mile uphill run equally to a 3.0 downhill mile run, even though the human experience of these is vastly different.

Labels

Labels that reveal the subjective experience of the runners are necessary to help the system to identify the features that are most likely to result in a fun run.

When deciding which examples, features, and labels are needed to train your ML model, work through your data needs on a conceptual level, as shown in the example below.

The example shows the data needs breakdown for a product that aims to solve the user need of “I want to fit runs into my busy schedule.”

Create a dataset specification

Just as you would create a product specification (such as a Product Requirements Document) before starting work on a new feature or product, consider writing a document to specify your data needs. Use the problem statement that you defined from the user needs to define the dataset that you will need.

For example, let’s say that you are working on developing the Plant Pal app to help users identify different types of plants and signal whether they are safe to touch and eat or not. You know that you’ll need to train an image classification model to do this, and you can start specifying the sort of the data that you’ll need to do this, the formats that you’ll need the data to be in (image formats, plant properties, and “safe to touch” and “safe to eat” labels), and potential data sources.

Get the data

Once you've defined the data requirements, you'll start gathering the data.

A good place to start is to determine if there are any existing datasets that you can reuse.

- Do you have access to existing datasets that meet your project requirements? Explore [Dataset Search](#) to start looking for available datasets.
- Can you acquire an existing dataset by partnering with another organization, purchasing a dataset, or using client data?

If you decide to acquire an existing dataset, make sure that you have the following information:

- Is this data appropriate for your users and use case?
- How was the data collected?
- Which transformations were applied to it?
- Do you need to augment it with additional data sources to be useful?
- Were any trade-offs and assumptions made when creating it?
- What are the data compliance standards and licensing information for the dataset?
- Does the dataset have any documentation, such as a [Data Card](#)?

Alternatively, or in addition to existing datasets, you might need to create a new dataset using an in-house data collection tool or a crowdsourcing platform.

Identify the data that you need vs. the data that you have

How much data and what type of data do you actually need for your project?

Sometimes, “more data” is not the best solution for your data needs issues.

Use the following questions to assess the data that you have as you determine what you need. For each of your variables:

- Does any of the data need to be treated differently? Some examples of such data include:
 - Personally Identifiable Information (PII)
 - Protected Characteristics
 - Variables that can be used to infer PII or Protected Characteristics.
- What benefit are you providing to the user by using this data?
- Can you store and use the data securely?
- How long will you keep the data?
- How much data do you actually need to label?
- Is the data representative of your users?
 - How would you define representativeness for your use case?
 - How would you collect representative data?

Balance underfitting & overfitting

To build products to work in one context, use datasets that are expected to reliably reflect that context. For example, for a natural language understanding model meant to work for speech, it wouldn't be helpful to use words that users type into a search engine as training data – people don't type searches the same way they talk.

If your training data isn't properly suited to the context, you also increase the risk of overfitting or underfitting your training set. Overfitting means the ML model is tailored too specifically to the training data, and it can stem from a variety of causes. If an ML model has overfit the training data, it can make great predictions on the training data but performs worse on the test set or when given new data.

Models can also make poor predictions due to underfitting, where a model hasn't properly captured the complexity of the relationships among the training dataset features and therefore can't make good predictions with training data or with new data.

There are many resources that can help the software engineers and research scientists on your team with understanding the nuances of training ML models so you can avoid overfitting and underfitting, for example these from Google AI. But first, involve everyone on your product team in a conceptual discussion about the examples, features, and labels that are likely required for a good training set. Then, talk about which features are likely to be most important based on user needs.

Commit to fairness

At every stage of development, human bias can be introduced into the ML model. Data is collected in the real world, from humans, and reflects their personal experiences and biases – and these patterns can be implicitly identified and amplified by the ML model.

While the guidebook provides some advice related to ML fairness, it is not an exhaustive resource on the topic. Addressing fairness in AI, and minimizing unfair bias, is an active

area of research. See Google's [Responsible AI Practices](#) for recent ML fairness guidance and recommended practices.

Here are some examples of how ML systems can fail users:

- **Representational harm**, when a system amplifies or reflects negative stereotypes about particular groups.
- **Opportunity denial**, when systems make predictions and decisions that have real-life consequences and lasting impacts on individuals' access to opportunities, resources, and overall quality of life. Take a look at [PAIR's Explorables](#) for a series of interactive essays that explore these ideas.
- **Disproportionate product failure**, when a product doesn't work or gives skewed outputs more frequently for certain groups of users.
- **Harm by disadvantage**, when a system infers disadvantageous associations between certain demographic characteristics and user behaviors or interests.

While there is no standard definition of fairness, and the fairness of your model may vary based on the situation, there are steps you can take to mitigate problematic biases in your dataset.

Use data that applies to different groups of users

Your training data should reflect the diversity and cultural context of the people who will use it. Use tools like [Facets](#) and [WIT](#) to explore your dataset and better understand its biases. In doing so, note that to properly train your model, you might need to collect data from equal proportions of different user groups that might not exist in equal proportions

in the real world. For example, to have speech recognition software work equally on all users in the United States, the training dataset might need to contain 50% of data from non-native English speakers even if they are a minority of the population.

Consider bias in the data collection and evaluation process

There is no such thing as truly neutral data. Even in a simple image, the equipment and lighting used shapes the outcome. Moreover, humans are involved with data collection and evaluation, and so, as with any human endeavor, their output will include human bias. See more in the section on [labeling](#) below.

For example, say you're creating a recommendation system to recommend new health and fitness goals to users. If the intent is to set goals that are safely achievable by users with a wide range of baseline fitness levels, it's important that the training dataset includes data from a variety of user types and not just young, healthy people.

For more on the topic of fairness, see Google's Machine Learning Fairness [Overview](#) and [Crash Course](#).

Manage privacy & security

As with any product, protecting user privacy and security is essential. Even in the running-related example above, the physiological and demographic data required to train this model could be considered sensitive.

Here are some suggestions for managing privacy and security:

- You may want to review the data for PII and Protected Characteristics
- You may want to consult with a lawyer before collecting or using such data in your region (and your product's users' regions).
- Don't assume basic data policies are enough to protect personal privacy.
- Set up the infrastructure, training and guidance programs for privacy protection and plan for situations where an adversary might get a hold of the data.
- Take extra steps to protect privacy (e.g., anonymize names, even if people agreed to have their name used) when personal details (e.g., addresses) could be exposed as part of AI predictions.


There are a number of important questions that arise due to the unique nature of AI and machine learning. Below are two such questions, but you should discuss these and others with privacy and security experts on your team.

What limits exist around user consent for data use?

When collecting data, a best practice, and a legal requirement in many countries, is to give users as much control as possible over what data the system can use and how data can be used. You may need to provide users the ability to opt out or delete their account. Ensure your system is built to accommodate this.


Is there a risk of inadvertently revealing user data? What would the consequences be?

For example, though an individual's health data might be private and secure, if an AI assistant reminds the user to take medication through a home smart speaker, this could partially reveal private medical data to others who might be in the room.



Local routes

- 1. Elevation matters** 12 mi
Steep hills and uneven terrain. Great for advanced runners.
■ Runner: "This one is a burner!"
- 2. Canal run** 2.3 mi
Beautiful scenery and a nice leisurely incline make this route unbeatable
■ Runner: "Wow, such a great view"
- 3. Rose path** 0.7 mi
This path goes right by the park. Be sure to stop and smell the roses.
■ Runner: "Perfect for pics"



Local routes

- 1. Elevation matters** 12 mi
Steep hills and uneven terrain. Great for advanced runners.
■ Diane Garza: "This one is a burner!"
- 2. Canal run** 2.3 mi
Beautiful scenery and a nice leisurely incline make this route unbeatable
■ Diane Garza: "Wow, such a great view"
- 3. Rose path** 0.7 mi
This path goes right by the park. Be sure to stop and smell the roses.
■ Diane Garza: "Perfect for pics"

Aim for

Take extra steps to protect privacy (anonymize names for example, even if people agreed to have their name used in community reviews) when personal details (such as where people live) could be exposed as part of AI recommendations or predictions. [Learn more](#)

Avoid

Don't assume basic data policies are enough to protect personal privacy. In this case, the runner agreed to expose her name in community reviews, but because she often starts runs from the same spot, another user could infer where she lives.

Key concept

Once your team has a high-level understanding of the data your product needs, work through your own translation between specific user needs and the data needed to produce them.

Try to be as specific as possible during this step. This will have a direct impact on which user experiences your team decides to spend your resources on going forward.

Apply the concepts from this section using Exercise 1 [in the worksheet](#)

③ Source your data responsibly

Use existing datasets

It may not be possible to build your dataset from scratch. As an alternative, you may need to use existing data from sources such as [Google Cloud AutoML](#), [Google Dataset Search](#), [Google AI datasets](#), or [Kaggle](#). If you're considering [supervised learning](#), this data may be pre-labeled or you may need to add labels (see more on [labeling](#), below).

Be sure to check the terms of use for the dataset and consider whether it's appropriate for your use case.

Before using an existing dataset, take the time to thoroughly explore it using tools like [Facets](#) to better understand any gaps or biases. Real data is often messy, so you should

expect to spend a fair amount of time cleaning it up. During this process, you may detect issues, such as missing values, misspellings, and incorrect formatting. For more information on data preparation techniques, check out the developer guidelines on [Data Preparation](#). They can help you make sure that this data will be able to help you deliver the user experience you identified at the outset.

Build your own dataset

When creating your own dataset, it's wise to start by observing someone who is an expert in the domain your product aims to serve — for example, watching an accountant analyze financial data, or a botanist classify plants. If you can interview them as they think or work through the non-ML solution to the problem, you may be able to pick up some insights into which data they look at when making a decision or before taking an action. Instead of one-off partnerships with domain experts, strive for tighter, ongoing collaborations and sustained relationships with domain experts throughout the project lifecycle.

You'll also want to research available datasets that seem relevant and evaluate the signals available in those datasets. You may need to combine data from multiple sources for your model to have enough information to learn.

Once you've gathered potential sources for your dataset, spend some time getting to know your data. You'll need to go through the following steps:

1. Identify your data source(s).

2. Review how often your data source(s) are refreshed.
3. Inspect the features' possible values, units, and data types.
4. Identify any outliers, and investigate whether they're actual outliers or due to errors in the data.

Understanding where your dataset came from and how it was collected will help you discover potential issues.

You may need to assemble a team of data collectors if you need to build a new dataset.

Document your data collection plan to help avoid quality issues, and review it for bias:

- Does the way you're asking a question affect the answer?
- Have you provided all of the necessary label options?
- Are you likely to get imbalanced classes? If so, consider over-sampling the rarer class.

Collect live data from users

Once your model has been trained and your product is being used in the real world, you can start collecting data in your product to continually improve your ML model. How you collect this data has a direct impact on its quality. Data can be collected implicitly in the background of user activity within your app or explicitly when you ask users directly.

There are different design considerations for each, which are covered in depth in the Feedback + Control chapter.

Capture the real world

Make sure that your input data is as similar as possible to real-world data to avoid model failure in production. Consider your user: do they have the time to take high-quality photographs, or will your model have to work with users' blurry smartphone images?

In many cases, you'll find that the data captured by users can be starkly different from that in your training dataset. Instead of striving for a very "clean" dataset that contains only very high-resolution images for an image classification problem, or only correctly formatted and typo-free movie reviews for a sentiment analysis problem, allow some 'noise' into your training dataset because you'll have plenty of it in the real world.

Consider formatting

Real data can be messy! A "zero" value could be an actual measured "0," or an indicator for a missing measurement. A "country" feature may contain entries in different formats, such as "US," "USA," and "United States."

While a human can spot the meaning just by looking at the data, an ML model learns better from data that is consistently formatted.

Avoid compounding errors from other ML models

If you're using the output of another ML system as an input feature to train your model, keep in mind that this is a risky data source. Errors associated with this feature will be

compounded with your system's overall error, and the further you are from the original training data, the more difficult it will be to identify error sources.

There's more information on determining error sources in the chapter on [Errors + Graceful Failure](#).

Protect personally identifiable information

No matter what data you're using, it's possible that it could contain personally identifiable information. Some approaches to anonymizing data include aggregation and redaction. However, even these approaches may not be able to completely anonymize your data in all circumstances, so consider consulting an expert.

[Aggregation](#) is the process of replacing unique values with a summary value. For example, you may replace a list of a user's maximum heartbeats per minute from every day of the month with a single value: their average beats per minute or a categorical high / medium / low label.

[Redaction](#) removes some data to create a less complete picture. Such anonymization approaches aim to reduce the number of features available for identifying a single user.

Prepare a data maintenance plan

If you want to create a dataset, consider whether you can maintain it and if you can afford the risks of something going unexpectedly wrong. Until the dataset is deprecated, focus on these tasks to maintain dataset quality:

- **Preventive maintenance:** Prevent problems before they occur. Store your dataset in a stable repository that provides different levels of access and stable identifiers. This can be difficult for datasets hosted on personal and lab websites.
- **Adaptive maintenance:** Preserve the dataset while the real world changes. Decide which properties of the dataset should be preserved, and keep this data updated over time. The dataset should still fulfill requirements, and continue to represent the reality of the phenomena that it measures.
- **Corrective maintenance:** Fix errors. Problems can occur as a result of data cascades. Make sure you have a plan B in case of unforeseen problems and human error. Keep a detailed, human-readable log of everything that you change in the dataset.

④ Prepare and document your data

Split your dataset into training and test sets

And finally, you'll need to split the data into training and test sets. The model is trained to learn from the training data, and then evaluated with the test data. Test sets are data that your model hasn't seen before – this is how you'll find out if, and how well, your model works. The split will depend on factors such as the number of examples in your dataset and the data distribution.

The training set needs to be large enough to successfully teach your model, and your test set should be large enough that you can adequately assess your model's performance. This is usually the time when developers realize that adequate data can

make or break the success of a model. So take the time to determine the most efficient split percentage. A typical split of your dataset could result in: 60% for training, and 40% for testing. [This lab](#) from Google AI offers more details on data splitting.

Analyze and prepare your data

An important step in all model training pipelines is handling “dirty” or inconsistent data.

Data cleaning often consists of two stages: detecting and addressing issues.

Examples of operations performed to clean data include:

- Extracting structure
- Dealing with missing values
- Removing duplicates
- Handling incorrect data
- Correcting values to fall within certain ranges
- Adjusting values to map to existing values in external data sources

Data cleaning and analysis require iteration. Analyzing and visualizing data may help identify issues in your dataset, possibly requiring further cleaning. Your team may need to return to this stage once you begin to evaluate your model, as previously undetected issues with your data may surface then, too. And if you find something unexpected in the data, consult with your domain expert.

To learn more about data preparation, see [Data Preparation and Feature Engineering in ML](#) and this [pre-ML checklist](#).

Document your data

Dataset documentation can be just as important as code documentation.

Having a record of your dataset's sources, a list of operations and transformations that have been applied to it, its history over time, and recommended uses can help when you do the following:

- Share your dataset with colleagues on your team or on another team
- Review whether you can use or publish your dataset
- Compare multiple datasets side by side
- Use the data responsibly in to train a model for an AI-powered product
- Interpret the behavior of an AI model that you trained with the data
- Maintain the dataset for teams and systems that rely on it

Data Cards are a form of documentation for datasets, and include information that can help answer the following questions about the data:

- What does it represent?
- What does it look like?
- Where does it come from?
- How was it prepared?
- Can it be used for a specific use case?
- How should it be used responsibly?

To learn more about how to create a Data Card to document your dataset, explore the [Data Cards Playbook](#).

Key concept

ML-driven products require a lot of data to work, and this is often where product teams falter. Getting enough data to both train and test your ML model is critical to delivering a functional product. To get you started, answer the key questions below:

- If you need to create a new dataset, how are you planning to collect the data?
- If you have an existing dataset, what, if any alterations or additions need to be made for your user population?

Apply the concepts from this section using Exercise 2 [in the worksheet](#)

⑤ Design for labelers & labeling

For supervised learning, accurate data labels are a crucial ingredient for achieving relevant ML output. Labels can be added through automated processes or by people known as labelers. “Labelers” is a generic term that covers a wide variety of contexts, skill sets, and levels of specialization. Labelers could be:

- **Your users:** providing “derived” labels within your product, for example through actions like tagging photos
- **Generalists:** adding labels to a wide variety of data through crowd-sourcing tools
- **Trained subject matter experts:** using specialized tools to label things like medical images

If people understand what you’re asking them to label, and why, and they have the tools to do so effectively, then they’re more likely to label the data correctly. Training your

labelers and testing their ability to complete the labeling task responsibly are both central to ensuring the quality of your data.

Ensure labeler pool diversity

Think about the perspectives and potential biases of the people in your pool, how to balance these with diversity, and how their points of view could impact the quality of the labels. In some cases, providing labelers with training to make them aware of unconscious bias has been effective in reducing biases.

Here are some questions to ask when evaluating your labeler pool:

- Are your annotators similar to your end users? E.g., are you asking American annotators to label images of Indian cuisine, or vice versa?
- Are there cultural differences between labelers and users that could impact data quality? E.g., date formatting rules (DD/MM/YYYY vs. MM/DD/YYYY) and measurement scales (metric vs. imperial) can differ.
- Do labelers know what to do?
 - Are they domain experts, or do they need training?
 - Are there guidelines on data quality, and have they been provided in the annotator's native language?
 - Do they know what the data will be used for?

Investigate labeler context and incentives

Think through the labeler experience, and how and why they are performing this task. There's always a risk that they might complete the task incorrectly due to issues like

boredom, repetition, or poor incentive design. Incentives that focus on volume rather than quality can lead to bias in your dataset.

Design tools for labeling

Tools for labeling can range from in-product prompts to specialized software. When soliciting labels in-product, make sure to design the UI in a way that makes it easy for users to provide correct information. When building tools for professional labelers, the article [First: Raters](#) offers some useful recommendations like the ones below:

- **Use multiple shortcuts to optimize key flows.** This helps labelers move fast and stay efficient.
- **Provide easy access to labels.** The full set of available labels should be visible and available to labelers for each item they are asked to address. It should be fast and easy in the UI to apply the labels.
- **Let labelers change their minds.** Labeling can be complicated. Offer a flexible workflow and support for editing and out-of-sequence changes so that labelers can seek second opinions and correct errors.
- **Auto-detect and display errors.** Make it easy to avoid accidental errors with checks and flags.
- **Review third-party tool capabilities and limitations.** Learn how labelers are onboarded, and if their training can be improved, and pilot the process on the platform to verify that labelers have a way to provide feedback and flag ambiguous data.

- **Explain criteria for acceptance.** Clearly state which errors can trigger task rejection.
- **Allow labelers to mark instances as ‘unsure’ or skip them altogether.** Don’t force labelers to label an item without providing checks and balances.
- **Value labeler disagreements.** Discovering discrepancies in how humans interpret labels makes model development more prepared for the real world, so make sure to revisit and review the data and labels.

When compiling instructions for labeler tasks, don’t take specialized domain knowledge for granted. Labelers may not be as familiar with the domain and the task, which leaves room for ambiguities.

Consider the following recommendations when writing instructions:

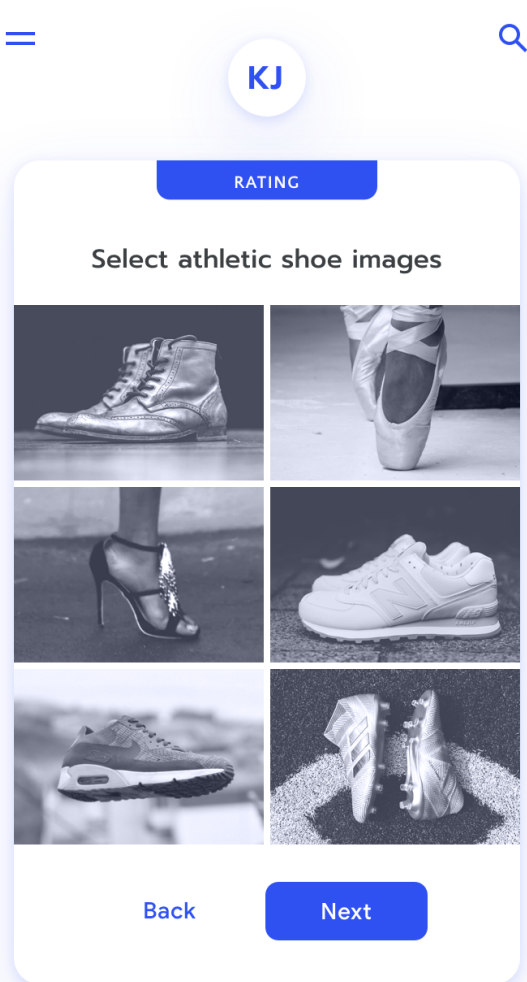
- **Provide examples for the task to illustrate expectations.** Include ‘edge’ cases.
- **Give step-by-step task instructions.** Explicitly describe all of the tools needed.
- **Break instructions down into manageable chunks.** Use bullets for steps, or rules.
- **Strike the right balance for task instruction length.** Compact tasks generally have greater uptake and turnaround time. However, longer task instructions are helpful when critical details are needed to perform the task accurately.

Plan ahead to conduct research with your labelers to iterate on task design, and instruction clarity. Before fully launching the task, pilot the task with a small set of labelers to gather their feedback and example results. You’ll also want to identify points of confusion directly from the labelers.

And finally, here are a few considerations for creating more accessible and inclusive labeler workflows:

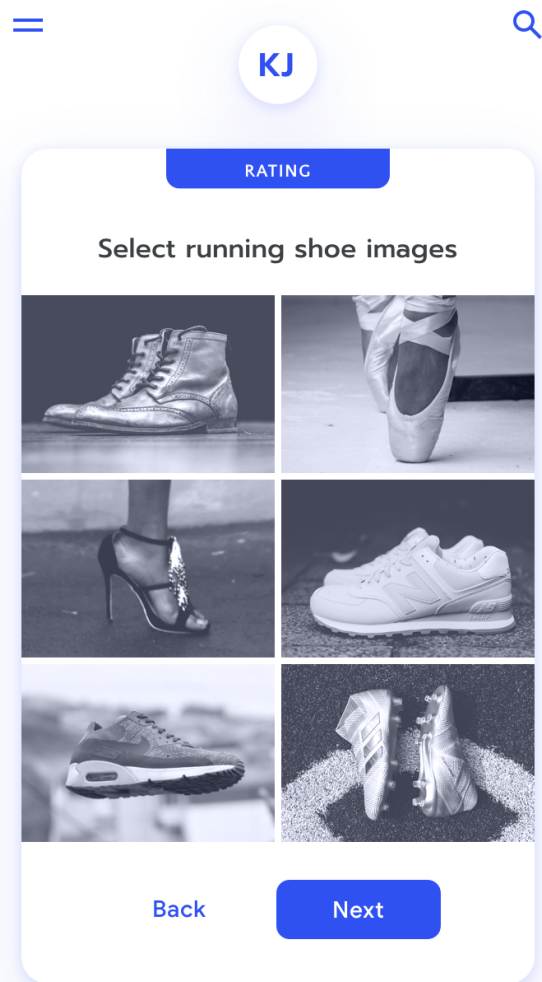
- **Account for worker environments, and the language of your data collaborators.**
You can help improve the representativeness of your data by designing tasks for your data collaborators.
- **Carefully experiment with the time allotment to ensure that labelers are able to complete the task at their own pace.** Make sure to give labelers enough time to complete tasks without rushing, and allow them to request extra time if needed.
- **Plan for mobile users in task design and time allotment.** Most traditional data labeling workflows are performed on laptops and desktops, but a growing online population is now 'mobile-first'. While this can open up opportunities for data collection from more diverse groups, such tasks can take longer, and require different task design for low-resource settings. Consider the following ways to adapt the workflow: minimize reliance on text and leverage other modalities (e.g., images and audio), create intuitive icons, and provide local language support.

Once you've collected data from labelers, you'll need to conduct statistical tests to analyze inter-labeler reliability. A lack of reliability could be a sign that you have poorly-designed instructions.



Aim for

Make labeler instructions as specific and simple as possible. Here, the phrase “running shoes” easily rules out the selection of other athletic shoe types like soccer cleats. [Learn more](#)



Avoid

Don't use instructions that can be interpreted multiple ways. Here, someone's subjective definition of “athletic” might or might not include dance shoes, for example.

Key concept

Before designing tools for your labelers, research their needs the same way you would think about your end-users. Their motivation and ability to do their job well has a direct impact on everything else you build down the line.

- Who are your labelers?
- What is their context and incentive?
- What tools are they using?

Apply the concepts from this section in Exercise 3 [in the worksheet](#)

⑥ Tune your model

Once your model has been trained with your training data, evaluate the output to assess whether it's addressing your target user need according to the success metrics you defined. If not, you'll need to tune it accordingly. Tuning can mean adjusting the hyperparameters of your training process, the parameters of the model or your reward function, or troubleshooting your training data.

To evaluate your model:

- Use tools like the What-If tool and the Language Interpretability Tool (LIT) to inspect your model and identify blindspots.
- Test, test, test on an ongoing basis.

- In early phases of development, get in-depth qualitative feedback with a diverse set of users from your target audience to find any “red flag” issues with your training dataset or your model tuning.
- As part of testing, ensure you’ve built appropriate and thoughtful mechanisms for user feedback. See more guidance for this in the [Feedback + Control](#) chapter.
- You may need to build custom dashboards and data visualizations to monitor user experience with your system.
- Be particularly careful to check for secondary effects that you may not have anticipated when determining your reward function, a concept covered in the [User Needs + Defining Success](#) chapter.
- Try to tie model changes to a clear metric of the subjective user experience like customer satisfaction, or how often users accept a model’s recommendations.

In addition to improving standard metrics (e.g., accuracy), you need a strategy for how to deal with a system that doesn’t behave as expected. Make sure to test your model before and after any changes. You may have to troubleshoot the training data in particular.

Consider the following:

- **Low data quality, or not enough high-quality data.** Can you go back and collect additional better data?
- **Unintended consequences.** These are opportunities for design changes. Look for:
 - Errors in output, and patterns in them
 - Changes in data that can lead to changes in model performance (data drift)

- Errors from the system itself, the context it's in or what the user wants
- How can you avoid these consequences in the next iteration?
- What other problems might arise next time & how do you mitigate them?
- **Parts of the project which are particularly difficult for users to understand.** A user might see something as an error if it's not explained correctly
 - Scan for omissions or white lies
 - Provide an explanation for these situations. E.g., "We use all your liked songs to generate recommendations, not just your favorite songs."

Once you've identified issues that need to be corrected, you'll need to map them back to specific data features and labels (or lack thereof), or model parameters. This may not be easy or straightforward. Resolving the problem could involve steps like adjusting the training data distribution, fixing a labeling issue or gathering more relevant data. Here's a hypothetical example of how a team might tackle tuning:

Let's say our running app was launching a new feature to calculate calories burned and make recommendations for mid-run changes to help users burn their target number of calories.

During beta testing, it was observed that users receiving these recommendations were far more likely than other users to quit mid-run. Moreover, users who followed the recommendations and completed the run and were less likely to return for a second run within the same week. The product manager originally assumed that this feature was a failure, but after user interviews and a deeper look at the data, it turned

out that the algorithm wasn't properly weighting important data when calculating estimated calorie burn like the outside temperature and a user's weight.

After some user research, it became clear that some users were quitting because they didn't trust the calorie calculation and therefore didn't see the point in accepting the app's recommendations for mid-run changes.

The engineering team was able to re-tune the algorithm and launch a successful feature.

Key concept

Tuning is an ongoing process for adjusting your ML model in response to user feedback and issues that arise due to unforeseen circumstances. Tuning never stops, but it is especially important in the early phases of development.

- What is our plan for doing early testing of our model?
- Is our set of early beta users diverse enough to properly test our model?
- What metrics will we use to determine if our tuning is successful?

Apply the concepts from this section on tuning by exploring the [What-if tool](#). Explore more about tuning models in response to user feedback in the [Feedback + Control](#) chapter.

Summary

Data is the bedrock of any ML system. Having responsibly sourced data, from a relevant context, checked for problematic bias will help you build better systems and therefore more effectively address user needs. Key considerations for data collection and evaluation:

- ① **Plan to gather high-quality data from the start.** Plan ahead as you gather and prepare data, to avoid the effects of poor data choices further downstream in the AI development cycle.
- ② **Translate user needs into data needs.** Think carefully as a cross-functional team about what features, labels, and examples you will need to train an effective AI model. Work systematically to break down user needs, user actions, and ML predictions into the necessary datasets. As you identify potential datasets, or formulate a plan to collect them, you'll need to be diligent about inspecting the data, identifying potential bias sources, and designing the data collection methods.
- ③ **Source your data responsibly.** As part of sourcing data, you'll need to consider relevance, fairness, privacy, and security. You can find more information in [Google's AI Principles](#) and [Responsible AI Practices](#). These apply whether you are using an existing dataset or building a new training dataset.

- ④ **Prepare and document your data.** Prepare your dataset for AI, and document its contents and the decisions that you made while gathering and processing the data.

- ⑤ **Design for labelers & labeling.** Correctly labeled data is a crucial ingredient to an effective supervised ML system. Thoughtful consideration of your labelers and the tools they'll be using will help ensure your labels are accurate.

- ⑥ **Tune your model.** Once you have a model, you will need to test and tune it rigorously. The tuning phase involves not only adjusting the parameters of your model, but also inspecting your data – in many cases, output errors can be traced to problems in your data.

Want to drive discussions, speed iteration, and avoid pitfalls? [Use the worksheet](#)