User Guide v25.7

See all our docs at docs.snorkel.ai

PDF generated on March 4, 2026

Welcome to Snorkel

Welcome to the Snorkel AI product documentation. Here you will learn about the Snorkel AI Data Development Platform and how to get started with a task.

What is a data development platform, and how can it help you create efficient and effective machine learning applications?

These articles introduce the product and key concepts:

- What is the Snorkel AI Data Development Platform?
- What is data-centric AI?
- Glossary

Get started with the Snorkel AI Data Development Platform

When you log in to the Snorkel AI Data Development Platform, you'll see the home page, which you can also navigate to by selecting **Home** or the **Snorkel** logo.

From the home page, you can see datasets, annotations, benchmarks, and evaluations. If you are a subject matter expert tasked with labeling data, you will see only the annotation batches assigned to you.

Deploy the platform

If you are an admin responsible for deploying and configuring the Snorkel AI Data Development Platform, read Installation Overview and Conventions.

Manage data

To get started, learn how to prepare and upload a dataset.

Label data

If you want to label or annotate data, read the Walkthrough for Annotators.

Evaluate an AI application

If you want to evaluate the performance of an ML application, explore the evaluation overview.

What is the Snorkel AI Data Development Platform?

The Snorkel AI Data Development Platform is a unified, data-centric solution designed to streamline the development of high-quality datasets and prompts for modern AI systems, including large language models (LLMs), retrieval-augmented generation (RAG) pipelines, and AI agents. By focusing on data and evaluation, the platform enables organizations to build specialized AI systems efficiently and effectively.

Snorkel's platform enables a structured, iterative development loop in which data scientists and domain experts:

- Annotate datasets to capture expert input
- Develop prompts to programmatically generate high-quality labels
- Apply slicing to identify subsets of data for targeted analysis
- Evaluate performance and conduct error analysis
- Improve data and prompts to close gaps and resolve errors

Each iteration through this loop improves the dataset and, in turn, the performance and trustworthiness of the downstream AI system.

What kinds of AI systems can I develop with Snorkel?

The platform supports data development for a broad range of use cases, including:

- RAG pipelines
- Agentic systems
- Structured extraction tasks
- Classification and triage systems
- Moderation, detection, and filtering tasks

These systems often require domain-specific data and evaluation criteria. Snorkel enables teams to encode and apply this knowledge programmatically to build and improve them efficiently.

What is data-centric development?

In contrast to trial-and-error prompt engineering or one-off fine-tuning jobs, data-centric development puts the data at the center of the AI system lifecycle. Rather than writing prompts or training models in isolation, you:

- Curate, label, and transform training and evaluation sets
- Develop task-specific labeling logic and prompt templates
- Evaluate results with targeted, interpretable feedback

This leads to systematic improvements and actionable insights across prompts, models, and pipelines.

What makes Snorkel different?

Snorkel gives you:

- **Programmatic labeling:** Quickly create high-quality labeled datasets using code, not just manual review.

- **Evaluation:** Analyze performance across interpretable subsets and failure modes.
- **Prompt development workflows:** Evaluate and improve generative systems in a structured, testable loop.
- **Expert-in-the-loop review:** Bring subject matter experts into the loop with workflows for error auditing and label correction.

The platform supports unstructured text, tabular, and semi-structured data. Whether you're processing customer tickets, contracts, user profiles, or financial reports, Snorkel helps you label, slice, and evaluate it more efficiently.

Snorkel replaces brittle, trial-and-error workflows with scalable, collaborative, and auditable systems for building and improving AI with data.

What is data-centric AI?

This document introduces data-centric artificial intelligence (AI) and how it differs from model-centric AI. It also explains how the data-centric approach can benefit machine learning (ML) applications and how it can be applied in an organization.

Data-centric AI vs. model-centric AI

In data-centric AI, data is the key factor in how well an ML application performs. A data-centric approach means your effort is weighted towards making sure AI is learning what you want it to learn. The Snorkel AI Data Development Platform makes this process of data development scalable with operations that accelerate labeling, managing, slicing, augmenting, and curating data efficiently. The model stays relatively fixed.

This contrasts with model-centric AI, where the choice of model is the differentiating factor. Model-centric AI uses training datasets as static collections of ground truth labels. The ML model is trained to fit that labeled training data as a static artifact.

Improvements in the data and improvements in the model are not mutually exclusive. Successful AI requires both well-conceived models and good data.

A brief history of data and models

Machine learning has always been about data. Historically, data science and machine learning teams would start with a dataset, and then achieve gains in ML application performance by iterating on static aspects of the data like feature engineering, or by working on algorithm design and bespoke model architecture.

Now that sophisticated models are available as off-the-shelf commodities, data science teams can provide value by shifting their focus towards continuously improving the data that makes their applications unique.

The power of iterating on data

Curated data shouldn't be treated as a static artifact.

Much like code forms the building blocks of traditional software, data forms the building blocks of ML applications. Just like you can iterate on code to improve traditional software, you can iterate on data to achieve higher performance in ML applications.

This holds true whether you're using your data in-context with prompt engineering and retrieval-augmented generation (RAG), or using data to fine-tune or train high-performance machine-learning (ML) models. If your focus is on fine-tuning and training models, today's models require significantly higher volumes of and higher quality training data.

Evaluating generative model responses is also a data task that benefits from iteration. For example, when you have a chatbot that can respond to user queries about your product, it may take several rounds of iteration to establish all of the criteria for an acceptable response, and then to label responses against those criteria.

Data-centric AI and the data development lifecycle

The dataset is the interface for collaboration between subject matter experts (SMEs) and data scientists, who turn those experts' knowledge into software.

The process of creating and improving curated datasets is called *data development*.

Just like software development, data development can be done programmatically, saving many hours of SME time. The Snorkel AI Data Development Platform enables data development as a discipline.

Example: Identify risk exposure using AI

To illustrate the value of data-centric AI, here's an example from one of Snorkel's customers.

Snorkel helped a top-three US bank reduce the time it takes to identify and triage risk exposure in their loan portfolio from months to days.

Previously, this process required:

- 6 person-months of analyst time to manually label the bank's documents
- 1 day to develop a model for the specific project
- 3 months to perform error analysis, validate the model, and review and improve the datasets

That six months of manual data labeling was a severe bottleneck. The bank was able to perform this risk exposure analysis only on an ad-hoc basis, and their datasets still suffered from inconsistent and unreliable labels. The labeling approaches weren't auditable and were slow to adapt to existing issues.

Before adopting the Snorkel AI Data Development Platform, the bank had these options for data labeling:

- Outsourcing to labeling vendors: Outside manual-labeling vendors brought significant privacy concerns regarding the confidentiality of the bank's data. Additionally, most annotators lacked domain expertise, so their work required additional review before use.
- Labeling with in-house experts: Using in-house subject matter experts (SMEs) solely for labeling was significantly time consuming and had a high opportunity cost, as SME time could be used on higher-value efforts.

When the bank decided to use the Snorkel AI Data Development Platform for data labeling, they significantly reduced total labeling time while increasing label accuracy:

- Increased speed: After developing their initial model in the Snorkel AI Data Development Platform, the bank could label ~250k new documents in less than 24 hours.
- Guided error analysis: Data scientists could rapidly iterate on model errors to improve results, collaborating with SMEs and codifying their input into the existing labeling approach.
- Minimal validation effort: Label validation and iteration could be accomplished in minutes with minor code modifications.

Key principles of data-centric AI

Here are some principles that can guide an organization or individual towards a data-centric approach to AI:

- Spend effort on data to efficiently improve ML applications: As models become more user-friendly and commoditized, gains in ML application quality increasingly rely on curated and updatable datasets.

- **Iterate on datasets:** Rather than treating your datasets as static inputs to ML applications, treat datasets like code and keep developing them.
- **Adopt programmatic data labeling:** Data development should be programmatic. This allows developers to cope with the volume of training data that today's deep-learning models require. Manually labeling millions of data points is not practical. A programmatic process for labeling and iterating on data is crucial to make consistent progress.
- **Collaborate with subject matter experts (SME):** Data-centric AI should treat subject-matter experts (SMEs) as integral to the development process. Include SMEs who understand how to label and curate your data so data scientists can inject their domain expertise directly into the model. Once done, this expert knowledge can be codified and deployed for programmatic supervision.

Benefits of data-centric AI

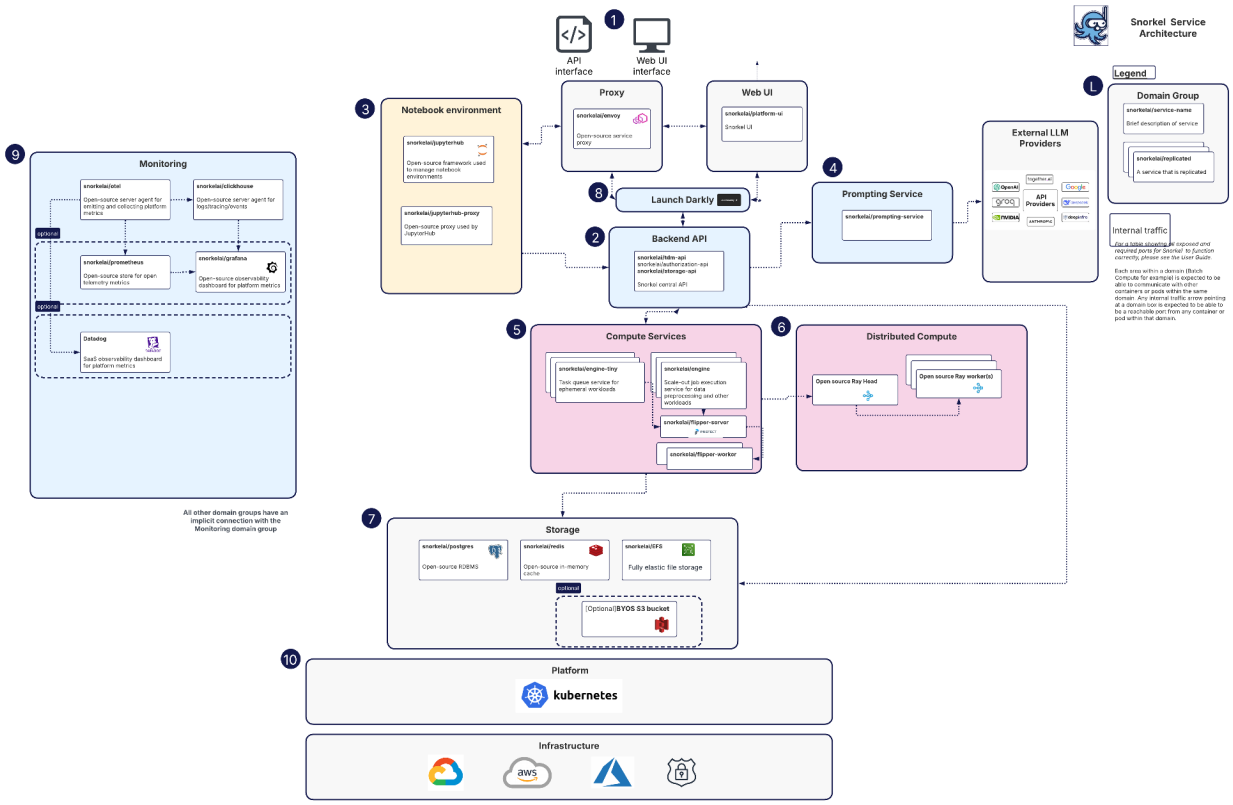
Organizations developing their AI solutions using data-centric AI capture additional value from creating higher-quality data:

- **Faster development:** A Fortune 50 bank built a news analytics application 45x faster and with +25% higher accuracy than a previous system.
- **Cost savings:** A large biotech firm saved an estimated $10 million on unstructured data extraction, achieving 99% accuracy using the Snorkel AI Data Development Platform.
- **Higher accuracy:** One of the largest custodian banks used the Snorkel AI Data Development Platform to extract and add metadata to a dataset. The bank improved the accuracy of their AI agent answers from 25% to 95% by using RAG from the enhanced dataset.

Snorkel architecture overview

The Snorkel AI Data Development Platform's architecture is comprised of the components illustrated below.

This document provides a detailed walkthrough of the platform's components, following the typical flow of data and requests through the system.



Select image to open full size

L: Legend

- Blue numbered circles (1–10) mark the typical request path and highlight key supporting systems, correlating directly with the numbered sections below.
- Each rounded rectangle on the diagram represents a **Domain Group**, a logical collection of related services or a distinct functional area within the platform.
- Cards contain a service name and description within a functional area.
- Stacked cards within a domain indicate replicated services for high availability and load balancing, or multiple instances of a component.

User interaction and control plane

1. User entry points & API gateway

Users interact with the Snorkel AI Data Development Platform through two entry points: the **Web UI** (`snorkelai/platform-ui`), designed for interactive workflows, and a programmatic **API**, accessible via

the SDK and notebooks.

All incoming traffic is routed through **Envoy** (`snorkelai/envoy`), which acts as the central API gateway. Envoy handles critical ingress functions, including TLS termination for secure communication, intelligent request routing to the appropriate backend services, and service discovery within the cluster.

2. Backend API and control plane

All user requests land on the **Backend API**, which functions as the platform's central control plane. This is a collection of core services responsible for orchestration and policy enforcement:

- `snorkelai/tdm-api`: Manages core application logic.
- `snorkelai/authorization-api`: Handles authentication (AuthN) and authorization (AuthZ).
- `snorkelai/storage-api`: Mediates access to the data plane.

3. Notebook environment

For users who require code-centric, exploratory workflows, the platform provides an integrated **JupyterHub environment**. This includes `snorkelai/jupyterhub` and `snorkelai/jupyterhub-proxy`, which are also fronted by Envoy to ensure consistent routing and security. Code executed within notebooks interacts with the same **Backend API** (Step 2) to ensure all operations are consistent and secure.

LLM and compute services

4. Prompting service & external LLMs

For workflows involving Large Language Models (LLMs), the **Backend API** includes prompting APIs that manage secure connections to various **External LLM Providers** like OpenAI, Anthropic, and Google.

5. Compute services and workflow orchestration

The Backend schedules all major data processing and model training workloads onto the platform's **Compute Services**. This layer is composed of:

- `snorkelai/engine-tiny`: Optimized for short-lived or ephemeral tasks.
- `snorkelai/engine`: Designed for scale-out, longer-running pipelines.

To manage complex, multi-step jobs, these engines use **Flipper** (`snorkelai/flipper-server` and `snorkelai/flipper-worker`), Snorkel's internal workflow orchestration service, to execute Directed Acyclic Graphs (DAGs) and other workloads.

6. Distributed execution on Ray

For tasks that require massive parallelism, Flipper orchestrates workloads on a **Ray Cluster** (composed of a Ray Head and Ray Workers). Ray provides a highly elastic and distributed compute environment, allowing the platform to efficiently scale processing across multiple nodes.

Data, storage, and platform foundations

7. Data plane and storage

All platform services and compute jobs interact with a robust and multi-faceted data plane for persistence and caching:

- **Postgres** (`snorkelai/postgres`): Serves as the primary relational database for application metadata, user information, and job state.
- **Redis** (`snorkelai/redis`): Provides a low-latency cache for frequently accessed data to accelerate performance.
- **EFS** (`snorkelai/EFS`): Offers shared elastic file storage for artifacts and datasets that need to be accessed across multiple services.
- **Bring-Your-Own (BYO) S3 Bucket (Optional)**: Customers can configure the platform to use their own S3-compatible object storage for full control over their datasets and model artifacts.

8. Feature flag management

To enable rapid iteration and safe rollouts, both client applications and backend services fetch feature flags from **LaunchDarkly**. This allows new features and workflows to be enabled or disabled dynamically without requiring a full redeployment of the platform.

9. Monitoring and observability

Comprehensive monitoring is a critical cross-cutting concern. Every service domain emits telemetry to the **Monitoring** group. The stack includes:

- **OpenTelemetry (OTel) agents** for collecting telemetry data.
- **Prometheus** for aggregating metrics.
- **ClickHouse** for storing logs and traces.
- **Grafana** for creating internal dashboards.
- **Datadog** is available as a SaaS dashboard option for customers.

10. Infrastructure on Kubernetes

The entire platform runs on **Kubernetes**, which handles container orchestration, scaling, and service management. This container-native approach ensures that the Snorkel Platform is infrastructure-agnostic and multi-cloud compatible, capable of being deployed on **Google Cloud (GCP)**, **Amazon Web Services (AWS)**, **Microsoft Azure**, or in on-premises data centers.

Internal traffic

Each area within a domain (Batch Compute for example) is expected to be able to communicate with other containers or pods within the same domain. Any internal traffic arrow pointing at a domain box is expected to be able to be a reachable port from any container or pod within that domain.

Evaluation Limits

When using Snorkel for evaluation, there is maximum of < 1k traces and < 100 steps per trace for each dataset.

LLM-as-a-judge (LLMAJ) iteration can only be used on train and valid splits.

You can only export the benchmark configuration via the Snorkel SDK.

Add Snorkel documentation to Cursor

Add the Snorkel documentation to Cursor to reference while you work. Follow these steps:

1. In the Cursor AI Pane, select **@ Add context > Docs**.
2. Scroll to the end of the list and select **+ Add new doc**.
3. In the **Add docs** text field, enter the Snorkel docs URL:
   - `https://docs.snorkel.ai/`
4. Verify the details in the text fields:
   - **NAME:** Enter `Snorkel` (or a name of your choice, like `Snorkel Docs`).
   - **PREFIX:** This should automatically populate with `https://docs.snorkel.ai/`.
   - **ENTRYPOINT:** This should also automatically populate with `https://docs.snorkel.ai/`.
   - **SHARE WITH TEAM:** Toggle this switch on if you wish to make this documentation source available to your team members within Cursor.
5. Select **Confirm**.
6. Start adding Snorkel docs context directly to queries within Cursor by typing `@Snorkel` followed by your question. For example:
   - **Example query:** `@Snorkel How do I add a new benchmark?`

   Note that if you chose a different name for the docs, you should use that name instead (for example, `@Snorkel Docs`).

You should now see chat responses in Cursor that draw on Snorkel's documentation, allowing for quick contextual access to information in your development environment.

Data preparation

Before uploading your data to the Snorkel AI Data Development Platform, we recommend that you go through the basic preparation steps outlined in this page (in addition to any custom preprocessing needed for your unique application) to minimize any unexpected data processing issues when managing your applications. After you are finished preparing your data, follow the steps in Data upload to upload your dataset to the Snorkel AI Data Development Platform.

> NOTE
>
> The examples on this page use data that is stored in Pandas DataFrame and the Snorkel AI Data Development Platform's object storage (MinIO). However, Snorkel supports data uploaded from other sources as well. See Supported data source types for more information.

Check for NaN, -Inf, and Inf values

We recommend removing `NaN`, `-Inf`, and `Inf` values from your data before uploading. *Out of range float values are not JSON-compliant* errors are common with these types of values while working with data in the Snorkel AI Data Development Platform.

There are many resources online that walk you through how to remove `NaN`, `-Inf`, and `Inf` values. Here is an example for sanitizing data in this format for Pandas DataFrames.

Check column types

We recommend checking the column types of your data to ensure that they are typed as expected. This can be done using `dtypes` for Pandas DataFrames.

> TIP
>
> The Snorkel AI Data Development Platform treats columns titled `url` as URLs, which means the raw URL and the content hosted at the URL is displayed. Snorkel renders the rich file format when you view this data for web pages, PDFs, images, and audio.

Make note of the label column name and label classes

If you have ground truth as a column in the data that you are uploading, make a note of the **label column** name and the **label classes**. You'll need to specify these while uploading data to the Snorkel AI Data Development Platform.

(Optional) split data into train, valid, and test sets

You can optionally split your data into **train**, **valid**, and **test** sets before uploading it to the Snorkel AI Data Development Platform. You many also choose to split by percentages during data upload to the Snorkel AI Data Development Platform.

- Train split: The train split does not require ground truth labels, but we recommend including a few ground truths so that you can utilize them during model development. You can also add ground truth labels in Snorkel's Annotation Studio after you upload the dataset.

- Valid split: The valid split should contain ground truth labels for evaluation purposes. This split can be used to tune your ML model.
- Test split: The held-out test split should contain ground truth labels for evaluation purposes.

The following example shows how data can be shuffled and split into train, valid, and test sets. The data is stratified over class labels in the column named `"label_column"`, with 70% of the data in the train set, and 15% in each of the valid and test sets.

```
# Shuffle dataset
df = df.sample(frac=1, random_state=42)
# 70% / 15% / 15% split by class
df_train = df.groupby("label_column").sample(frac=0.7, random_state=42)
df_valid = df.drop(df_train.index).groupby("label_column").sample(frac=0.5,
random_state=42)
df_test = df.drop(df_train.index).drop(df_valid.index)
```

> **TIP**
> Be sure to check out our Tips for splitting and partitioning data guide when preparing data splits.

Export the data splits as Parquet or CSV files

If you are uploading data from a Pandas DataFrame via object storage or locally, first export the data to a Parquet or CSV file.

Snorkel recommends Parquet files instead of CSV files to avoid column type ambiguities. For example, defining if "123" is meant to be ingested as a `string` or `integer` type. Parquets explicitly store these column types; CSVs do not.

The following example shows how to export data from a Pandas DataFrame to Parquet files:

```
# Export to Parquet file
df_train.to_parquet("/path/to/my/dataset_name_train.parquet")
df_valid.to_parquet("/path/to/my/dataset_name_valid.parquet")
df_test.to_parquet("/path/to/my/dataset_name_test.parquet")
```

(Optional) Upload data to Snorkel's object storage (MinIO)

Data can be uploaded from a variety of sources (see Supported data source types for more information). One option is to upload data from the MinIO object storage system deployment that comes with the Snorkel AI Data Development Platform. This can be a good option if your data is stored locally and is larger than 100MB.

If you plan to use MinIO as a data source, you need to upload it to MinIO first. Follow the steps below:

1. **Access Snorkel's object storage**. On the bottom of the left-side menu, click your user name, then click **Resources → MinIO object storage**. From there you can login to the MinIO console. For any issues with

the access key, or for access issues with Kubernetes installations, contact your Snorkel administrator.

2. **Create a bucket to upload your data**. On the bottom right corner of your screen, click the **red +** button, then click **Create bucket**. From there, you can name your bucket.

   > NOTE
   > Make sure that you use lowercase characters while naming your bucket. Using upper-case characters will result in an error.

   > NOTE
   > MinIO does not create a folder at the specified location until a file is uploaded. If you plan to use a bucket programmatically and need the folder to exist, upload an empty file in the UI to create the folder.

3. **Upload your data**. Once your bucket is created, click the **red +** button again, then click **Upload file**. Select the files that you'd like to upload.

Make sure that you remember your file paths on MinIO, as you'll need them to create a data source on the Snorkel AI Data Development Platform. The file path is in the form `minio://<bucket>/<path/to/file>`. For example, a file named `train.parquet` inside the bucket `mybucket` in MinIO would be referred to as `minio://mybucket/train.parquet`.

Supported data formats

The Snorkel AI Data Development Platform does not support the DateTime data format. When you import data to the Snorkel AI Data Development Platform, convert DateTime to String.

When you save a Snorkel DataFrame to a `.csv` file, Snorkel automatically converts unsupported data formats to supported data formats. For example, Snorkel converts DateTime to String. If you import a `.csv` generated from another source, you must convert unsupported data types manually before importing the file to Snorkel.

Supported data source types

This page provides information about the data source types that the Snorkel AI Data Development Platform supports. You can follow the steps outlined in Data preparation and Data upload to prepare and upload your dataset into the Snorkel AI Data Development Platform.

The Snorkel AI Data Development Platform supports the following types of data sources: Amazon S3, Google Cloud Storage (GCS), local files, SQL, Databricks SQL, Google BigQuery, and Snowflake.

> NOTE
> When you use credentials to access your data, your credentials are encrypted, stored, and managed by Snorkel. Credentials are not logged or stored unencrypted.

Amazon S3

Select **Amazon S3** to add data from private or public S3 buckets. Turn on the "Use credentials" toggle to use data in private buckets.

Fill out the required fields:

- **Access key ID**: Your AWS access key ID
- **Access key**: Your AWS Secret Access key
- **Region**: AWS region

Google Cloud Storage

Select **Google Cloud Storage** to add data from your GCS. Turn on the "Use credentials" toggle to use data in private buckets.

Fill out the required fields:

- **Project ID**: Your Google Cloud project ID.
- **JSON service account file contents**: The raw JSON contents of a key file that belongs to a service account with access to Google BigQuery. Please note that the service account requires `roles/bigquery.readSessionUser` and `roles/bigquery.dataViewer` to read a table.

Local file

Select **File Upload** to add data sources from your local machine. You must upload a Parquet or CSV file.

> NOTE
> Local data upload currently only supports data source files up to a maximum size of 100MB. If your data files are too large, use a cloud storage service like AWS S3 or Google Cloud Storage.

SQL

Select **SQL DB** to add data sources from queries against SQL databases like Postgres and SQLite.

Fill out the required fields:

- **Connection URI**: A database connection string, for example, those that are used by SQLAlchemy.
- **SQL query**: A SQL query where each result row is a data point.

Databricks SQL

Select **Databricks SQL** to add data sources from queries against a Databricks SQL warehouse. Currently, all table columns are read.

Fill out the required fields:

- **Server hostname**: The server hostname of your Databricks SQL warehouse.
- **HTTP path**: The HTTP path of your Databricks SQL warehouse.
- **Access token**: Your Databricks access token.
- **SQL query**: A Databricks query where each result row is a data point.

See the Databricks documentation for more details about these items.

Note
Your credentials are encrypted, stored, and managed by the Snorkel AI Data Development Platform. Credentials are not logged or stored unencrypted.

Google BigQuery

Select **Google BigQuery** to add data sources from Google BigQuery tables. Currently, all table columns are read.

Fill out the required fields:

- **Project ID**: Your Google Cloud project ID.
- **JSON service account file contents**: The raw JSON contents of a key file that belongs to a service account with access to Google BigQuery. Please note that the service account requires `roles/bigquery.readSessionUser` and `roles/bigquery.dataViewer` to read a table.
- **Google BigQuery table specification**: A JSON specification for the table columns to read, where each result row is a data point. The specification must have the following keys, and an example is provided below.
  - `dataset_id`: The BigQuery dataset ID.
  - `table_id`: The BigQuery table ID.
  - `columns`: The list of columns to include.

```
{
 "dataset_id": "bigquery-public-data.noaa_tsunami",
 "table_id": "historical_source_event",
 "columns": ["id", "year", "location_name"]
}
```

Snowflake

Select **Snowflake** to add data sources from queries against a Snowflake data warehouse.

Fill out the required fields:

- **Username**: Your Snowflake username.
- **Account identifier**: Your Snowflake account identifier.
- **Password:** Your Snowflake password.
- **Snowflake query**: A Snowflake query where each result row is a data point. Queries can specify the database, schema, and table, as in the example below.

```
SELECT c_name, c_address
FROM snowflake_sample_data.tpch_sf1.customer
LIMIT 10;
```

> NOTE
>
> The default warehouse that is specified for your Snowflake account will be used.

Upload ground truth

This topic demonstrates how to upload ground truth (GT) in the Snorkel AI Data Development Platform. There are two types of GT in the Snorkel AI Data Development Platform: document-level GT and span-level GT.

Document-level ground truth

Document-level GT applies when each document has one label.

Uploading from a dataset

In a Dataset, navigate to the **Data Sources** tab. From there, click on **Upload ground truth** button on the right. Add your ground truth file for upload.

> NOTE
>
> Only classification label schemas are supported for upload on the **Datasets** page.

Data format

In particular, the GT contains the following columns:

- `uid` (int): The uid of the document.
- `label` (str): The GT label of the document.

Span-level ground truth

Span-level GT applies when each span has one label.

Data format

The uploaded span-level GT must contain the following columns:

- `context_uid` (int): The UID of the document from which the span was extracted. All spans that have the same `context_uid` come from the same document. The `context_uid` is associated with the **UID column** when one creates the dataset. For more, see Data upload.
- `span_field` (str): The field where the spans were extracted from.
- `char_start` (int): The index of the first character of the span in the document.
- `char_end` (int): The index of the last character of the span in the document.
- `_gt_label` (str): The GT label of the span.

As an example, below is an example ground truth for a single document with `context_uid = 0`, and extraction field of `text`:

gt_label	context_uid	span_field	char_start	char_end
class_1	0	text	17	28
class_2	0	text	37	46

> NOTE
>
> Only the spans already extracted by the candidate extractor can be recognized as GT. Both the span extracted and the span in the GT must have the same vaule of `context_uid`, `char_start` and `char_end`.

We recommend you to iterate on the extractor until you get as close as possible to 100% recall. One can evaluate the performance of the candidate extractor via `sf.get_candidate_extractor_metrics` SDK method.

If you find most of your GT spans overlap with the extracted spans, you can consider resolving you GT to be exactly match with the extracted spans. For example, in the sentence `It's a lovely day.`, if the GT span is `day`, but the extracted span is `lovely day`, we can resolve the GT span to be `lovely day` and use the corresponding `char_start` and `char_end` of `lovely day` in the span level GT.

Uploading a dataset

The Snorkel AI Data Development Platform organizes data into **data sources** and **datasets:**

- **Data sources** are individual partitions of data points, such as the rows of an individual parquet file or resulting rows from an individual SQL query. Each data source is assigned to exactly one split (train, valid, or test).
- **Datasets** are collections of data sources, with each data source assigned to a specific split (train, valid, or test) within the dataset.

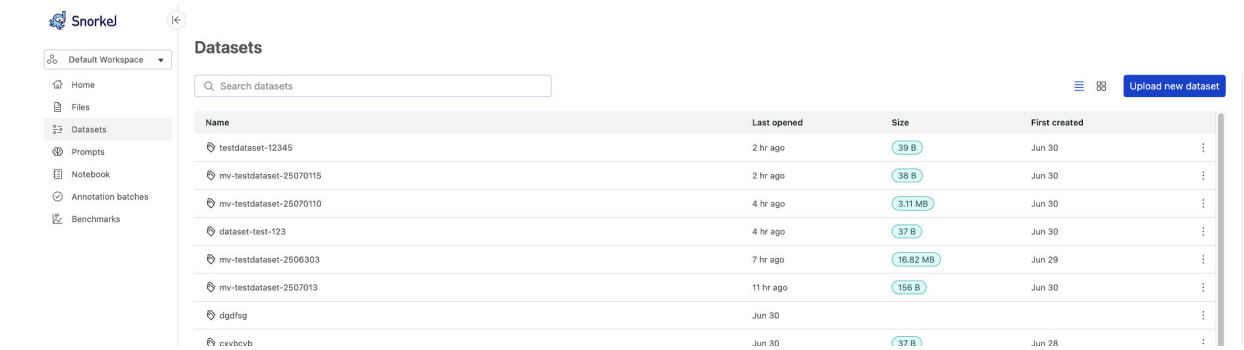
You can upload datasets to the Snorkel AI Data Development Platform, starting with a single data source. Data sources can be added to datasets at any time.

Prerequisite

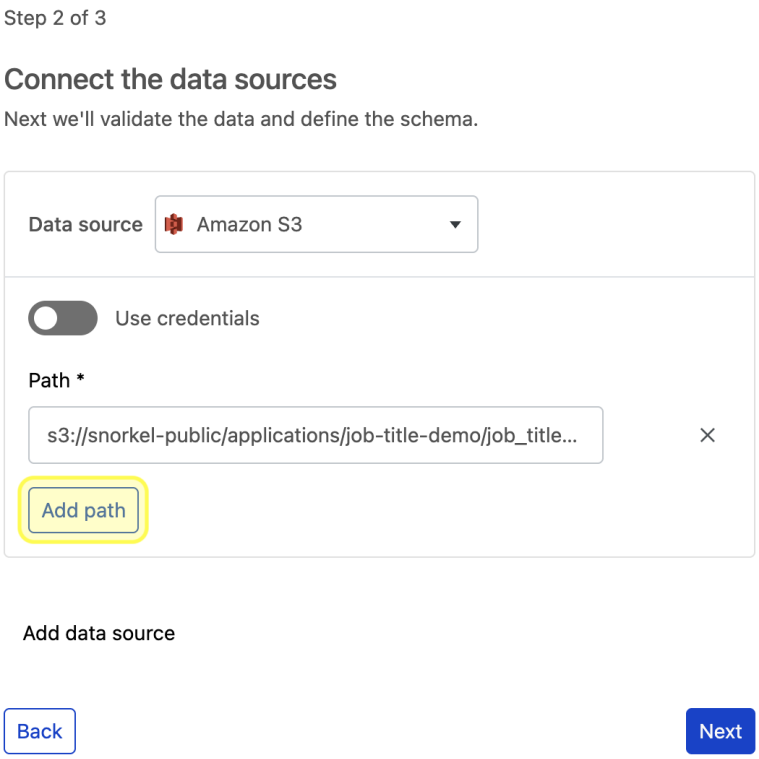
Before uploading your dataset to the Snorkel AI Data Development Platform, prepare your data to minimize any unexpected data processing issues when managing your applications.

To upload a dataset

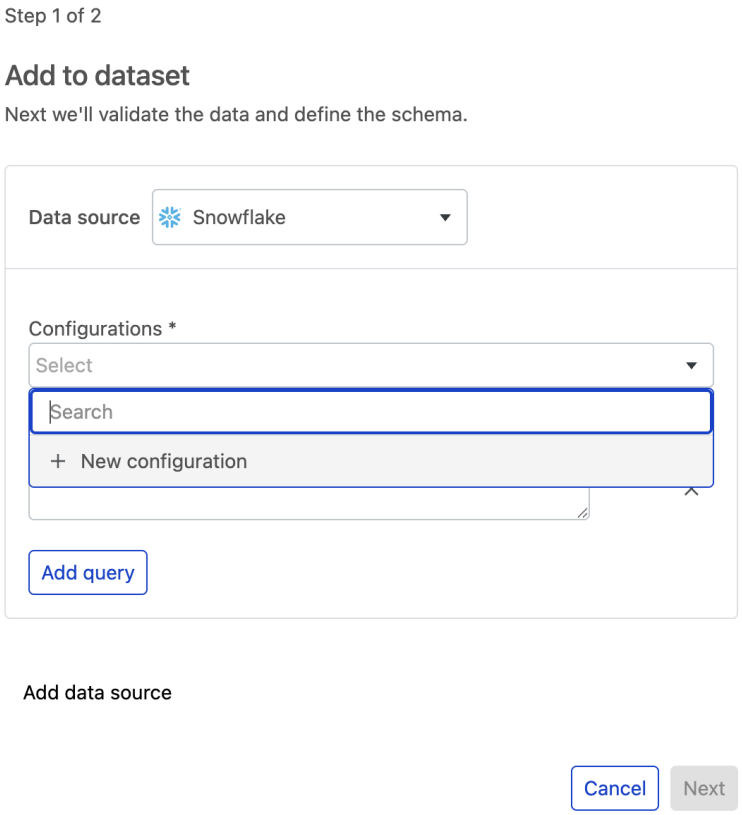
1. Select the **Datasets** option in the left navigation, and then select **Upload new dataset** in the top right corner.



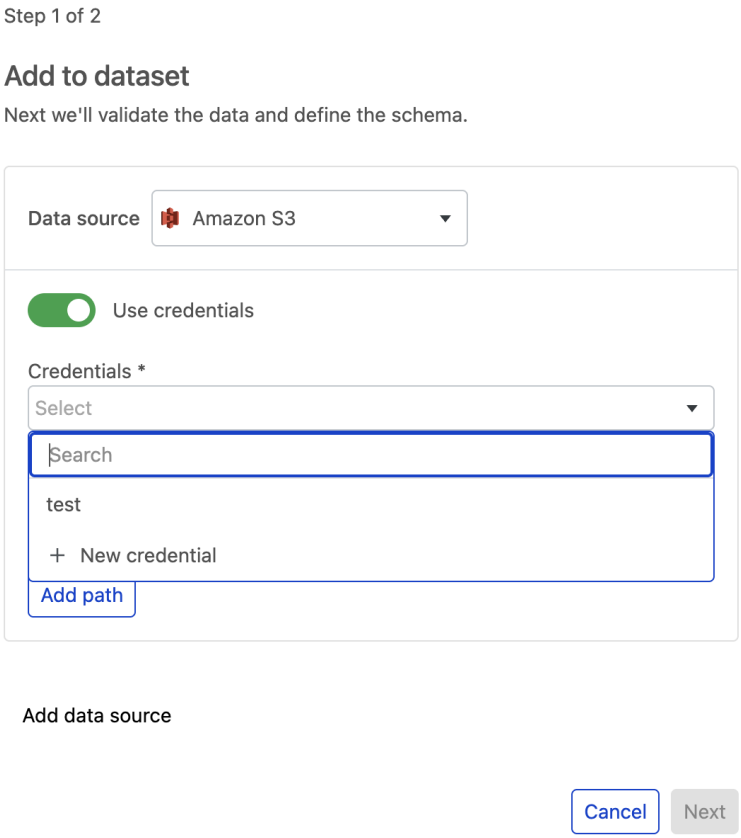
2. For the **New dataset**, enter a **Dataset name**. Select **Next**.
3. Select you **Data source** from the dropdown menu. Learn more about supported data source types.
4. Put in the path or query for the chosen data source. To ingest multiple paths or queries for that data source, select **Add path** or **Add query**, based on your chosen data source type.



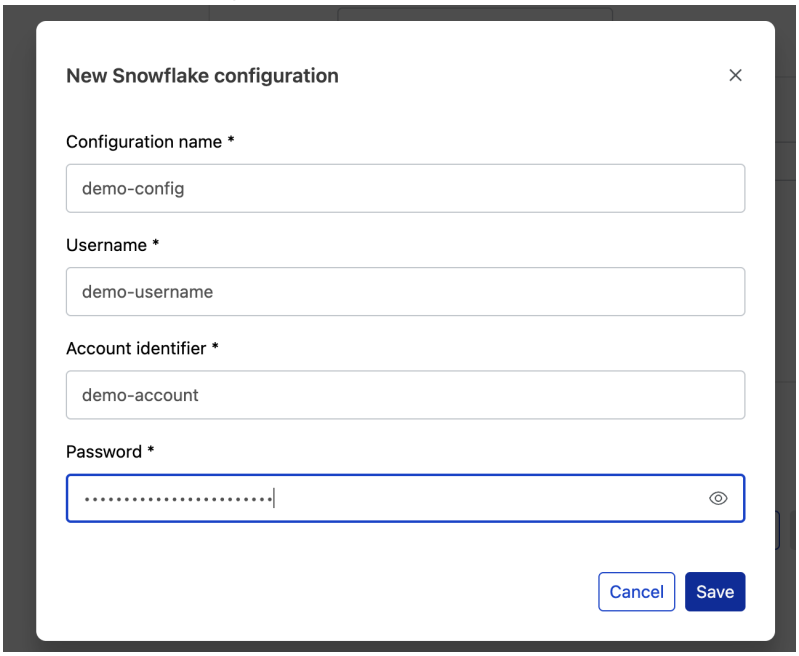
a. Alternatively, you can also use credentials to ingest data from private data sources. This is mandatory for some data source types, such as Databricks and Snowflake. This is optional for other sources, such as S3 buckets, which may be public or private.



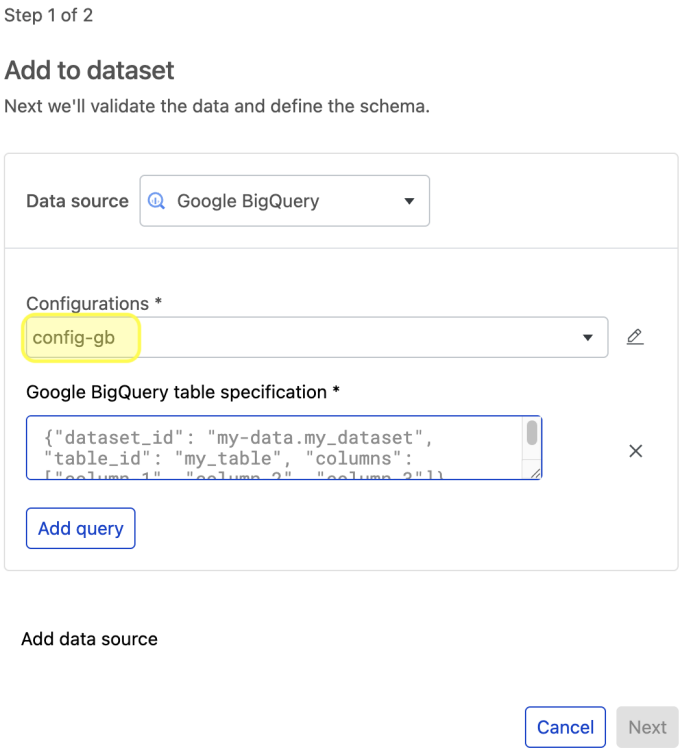
b. To ingest from a private S3 bucket, toggle on **Use credentials** and select **+ New credential**.



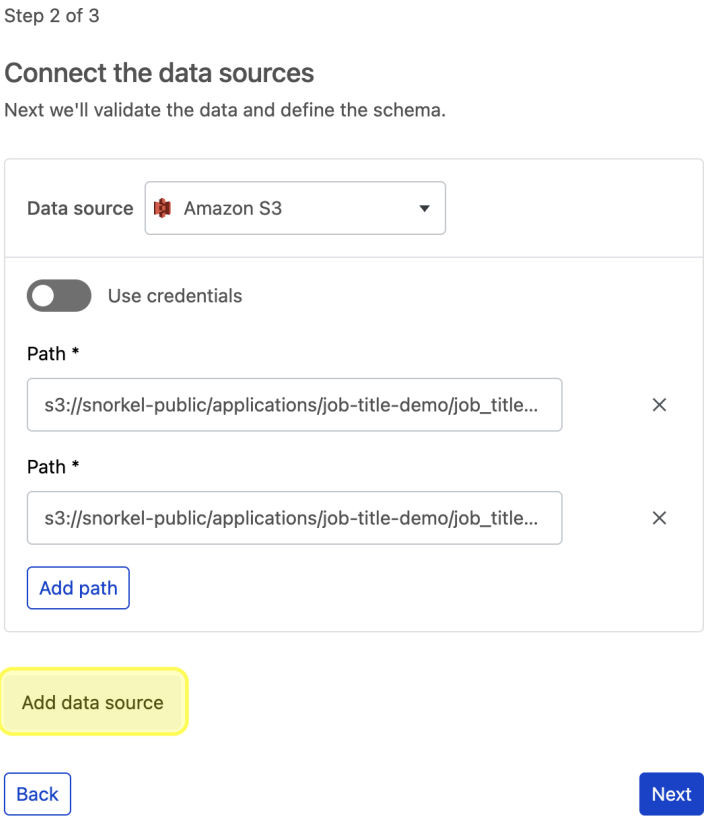
In the module, add your credentials and select **Save**.



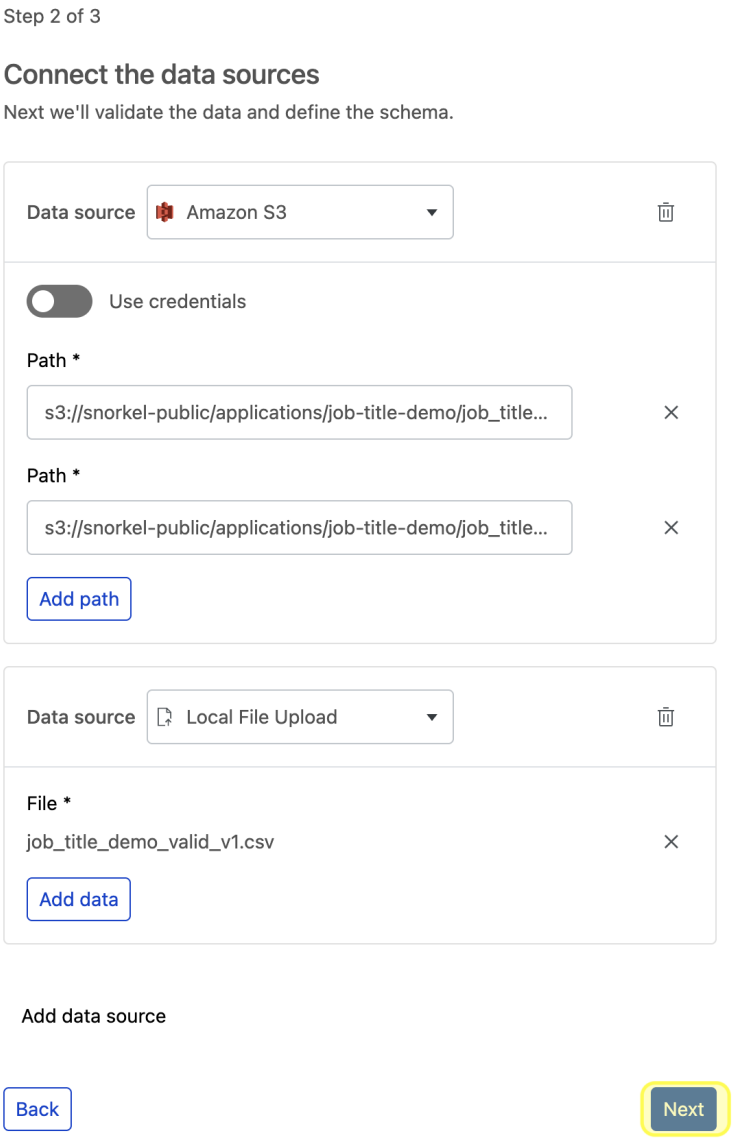
Select your saved configuration from the dropdown menu and enter your path or query to ingest data from the new configuration.



5. To add multiple data sources with one or more paths or queries under each, select **Add data source**.

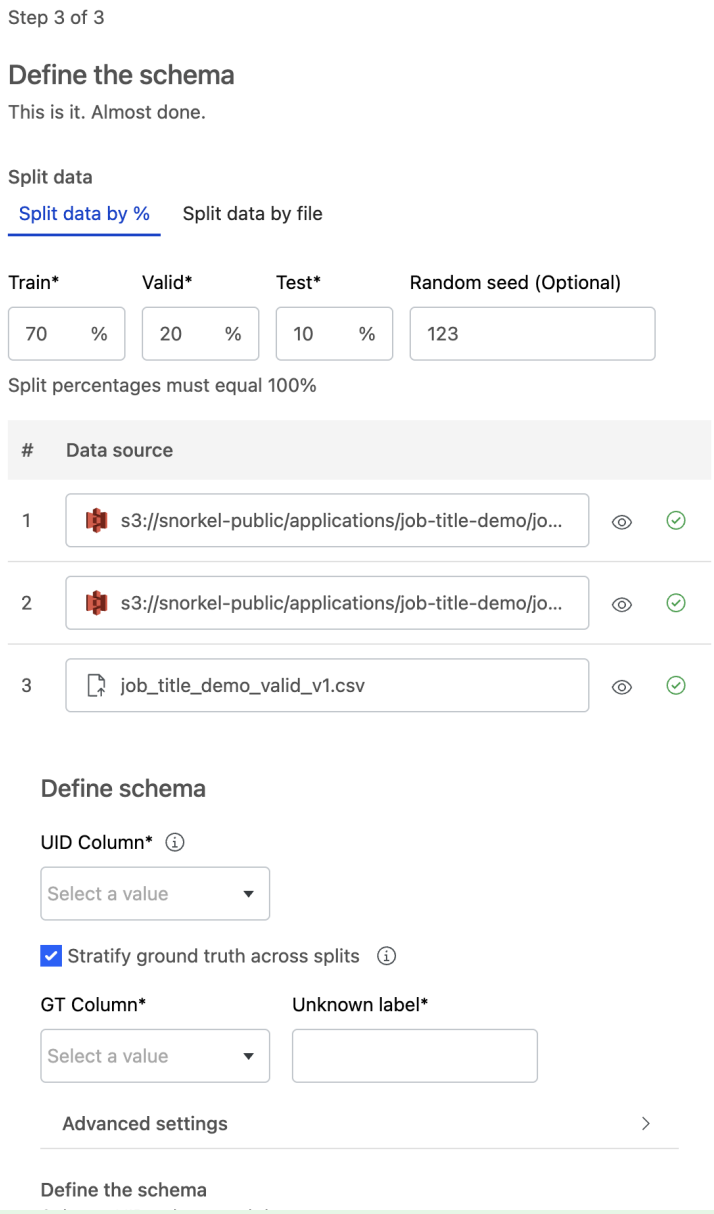


6. Repeat these steps for each data source, and then select **Next** to verify your data sources. This step verifies that your data sources are accessible and runs basic checks, such as detecting `NaN` values, to ensure your data is valid for ingestion.



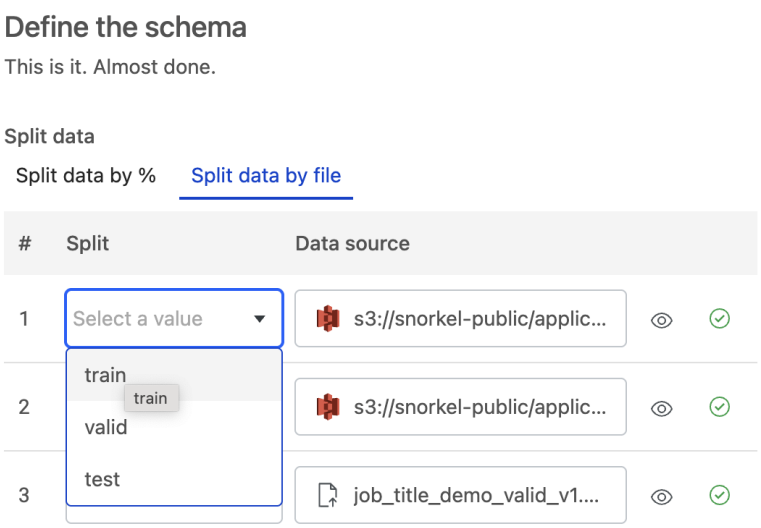
7. Assign splits to data sources. There are multiple ways to assign splits to data sources:

   - Automatically with the **Split data by %** option. If you select this option, then you need to define your ground truth (GT) column.

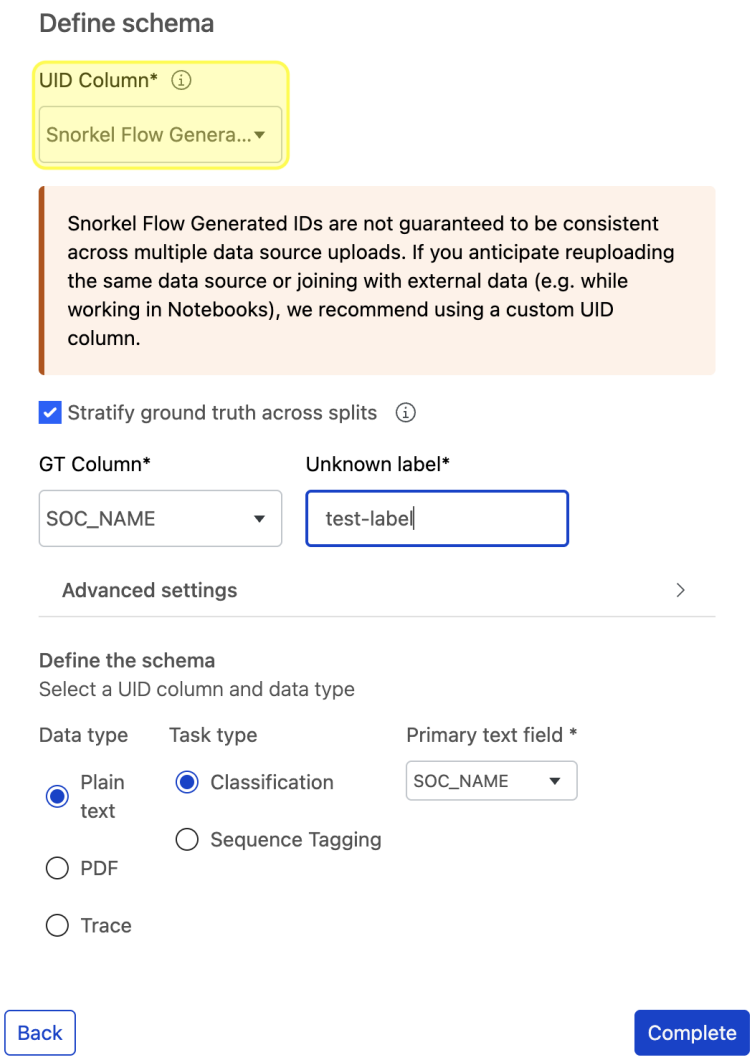


> **TIP**
> If you have a large amount of unlabeled data, split your training data into smaller data sources. You can enable a subset of data sources for faster initial development before scaling up to all of your data.

- Manually with the **Split data by file** option.



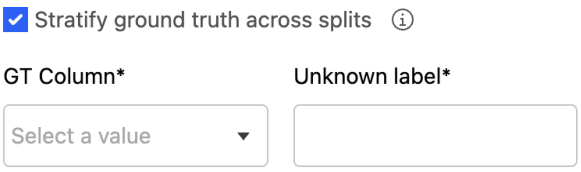
8. Once all of your data sources are verified, choose a **UID column** that is unique for each row across the dataset. This UID column can be text or an integer. If your dataset does not have that type of field, then choose **Snorkel Generated** to have the Snorkel AI Data Development Platform generate a UID column.



Once the dataset is created, a new `context_uid` column is added to your data. This column is populated with the selected **UID column** or the **Snorkel Generated** UID.

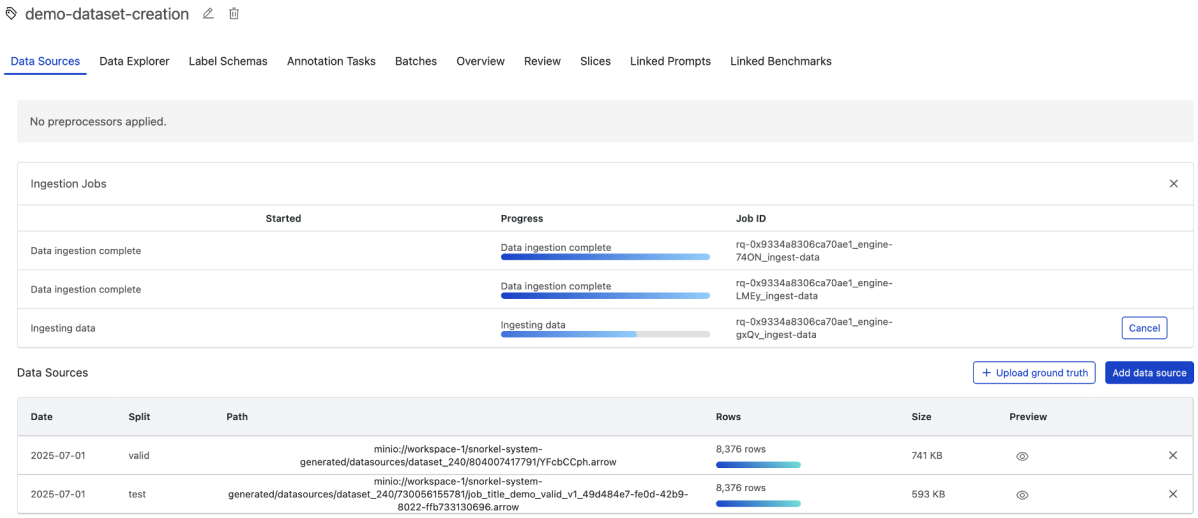
a. If you chose to automatically split your data with the **Split data by %** option, you can **Stratify ground truth across splits** to ensure the ground truth labels are evenly distributed across the splits.

- If you opt in to stratify ground truth, provide the **GT column** and the value corresponding to UNKNOWN ground truth.
- If you opt out, the data is split randomly.



b. Fill in the data type for the task.

9. Select **Complete** to begin ingesting your data into the platform.



Managing files in data storage

User files are the assets related to data points in a given data source. These files can be associated with your data sources.

Permissions

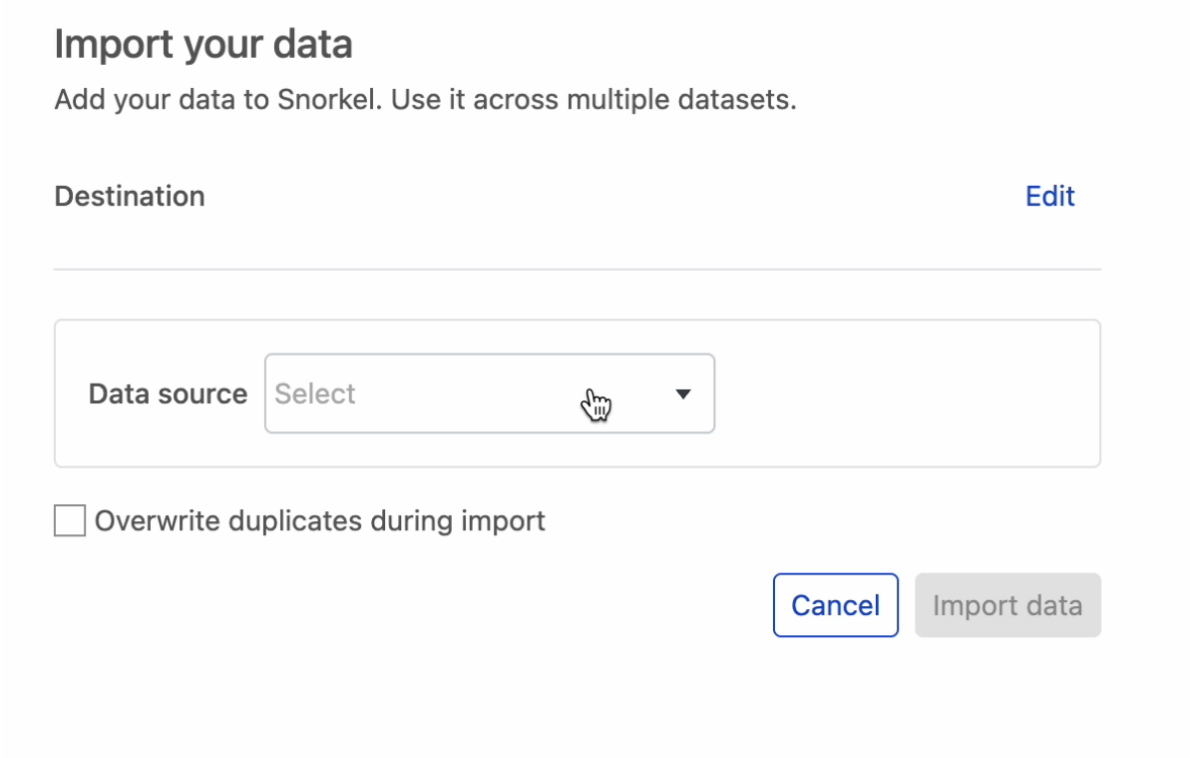
To upload user files, ensure you have these instance-level permissions:

- Direct file uploads from the computer that is running the Snorkel AI Data Development Platform.
- Remote file transfer from a cloud storage provider, such as Amazon S3 or Google Cloud Storage.

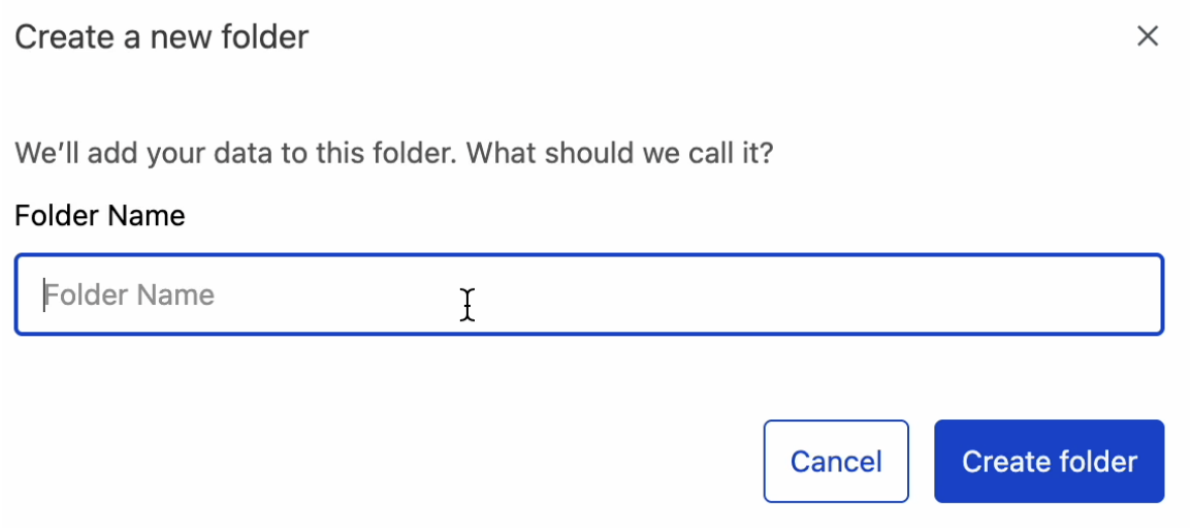
An administrator may disable upload permissions for the instance you are working on, which could prevent you from uploading or transferring user files.

To upload user files

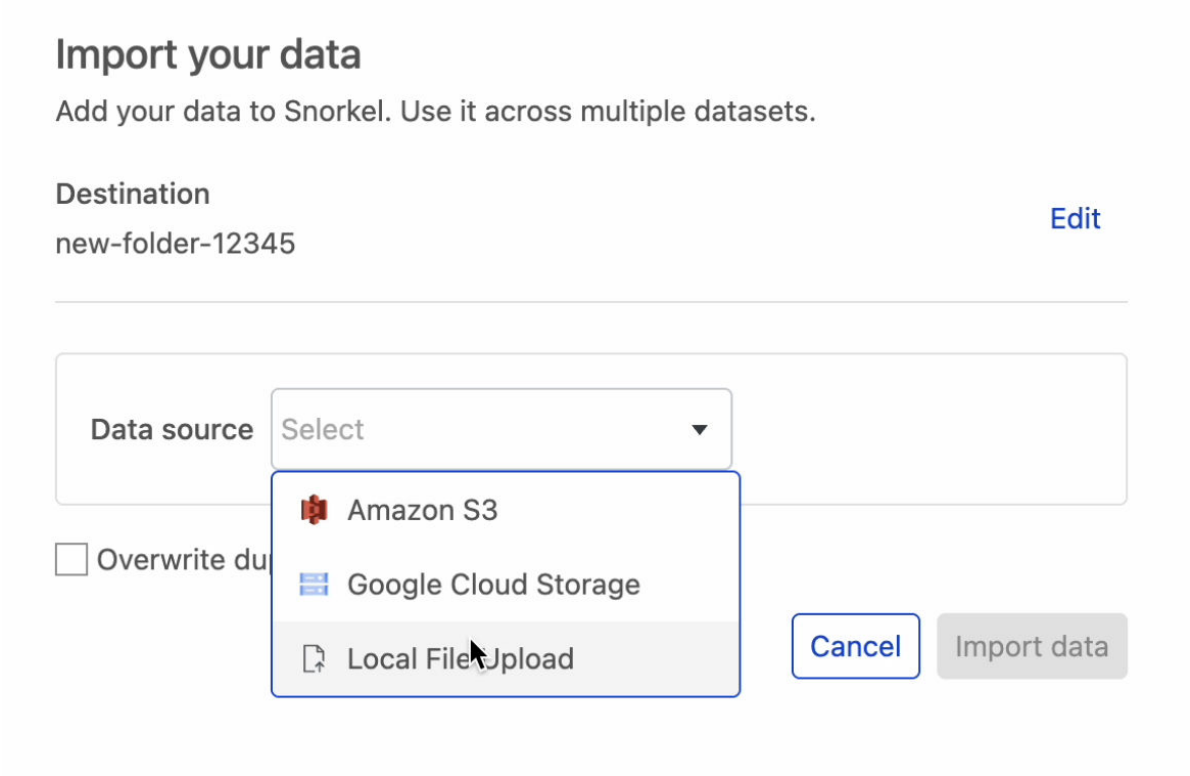
1. To upload user files, select **Data Storage**.
2. Select the **Import data** button.



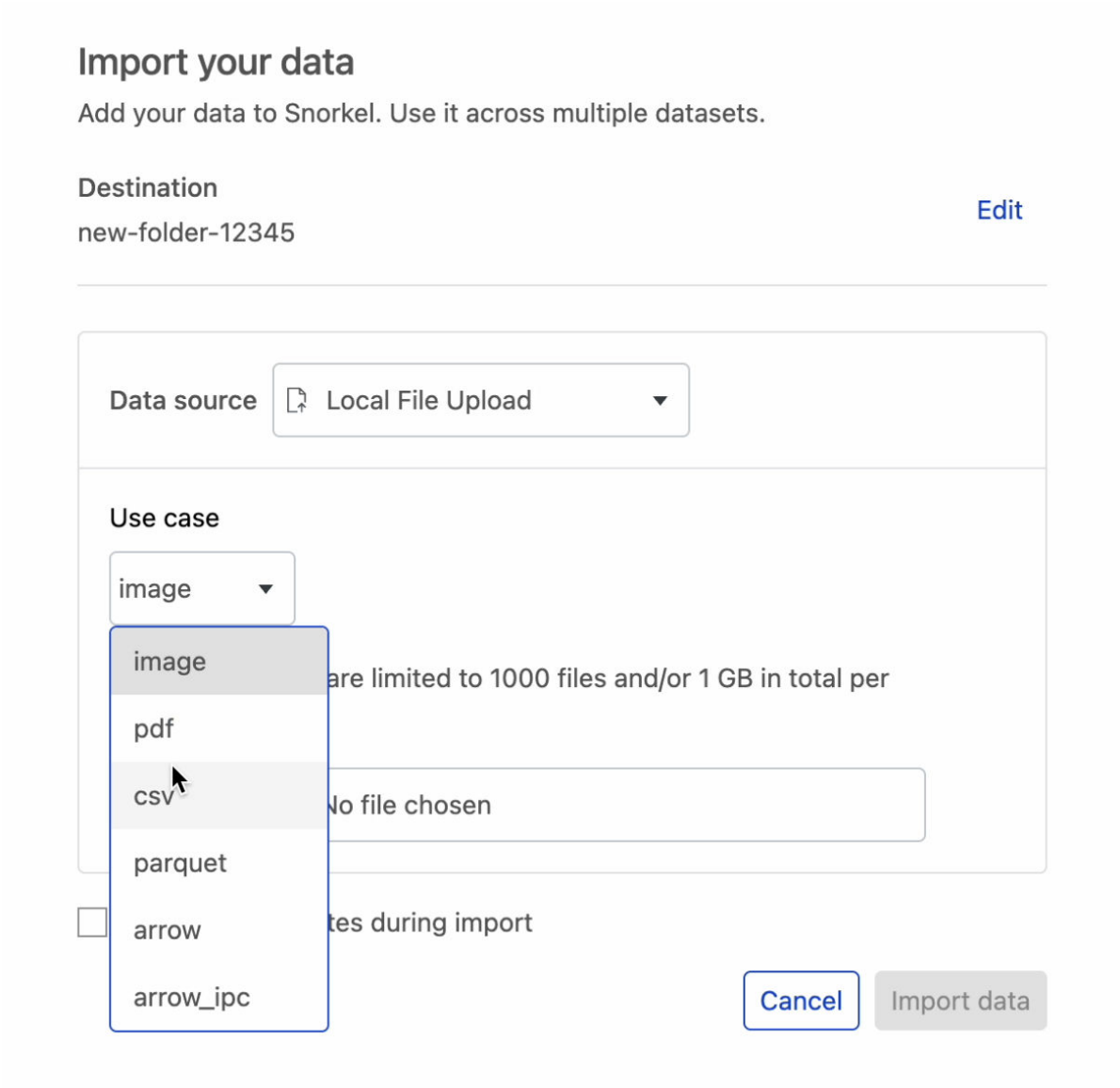
3. Choose an existing folder from the dropdown or create a new folder.



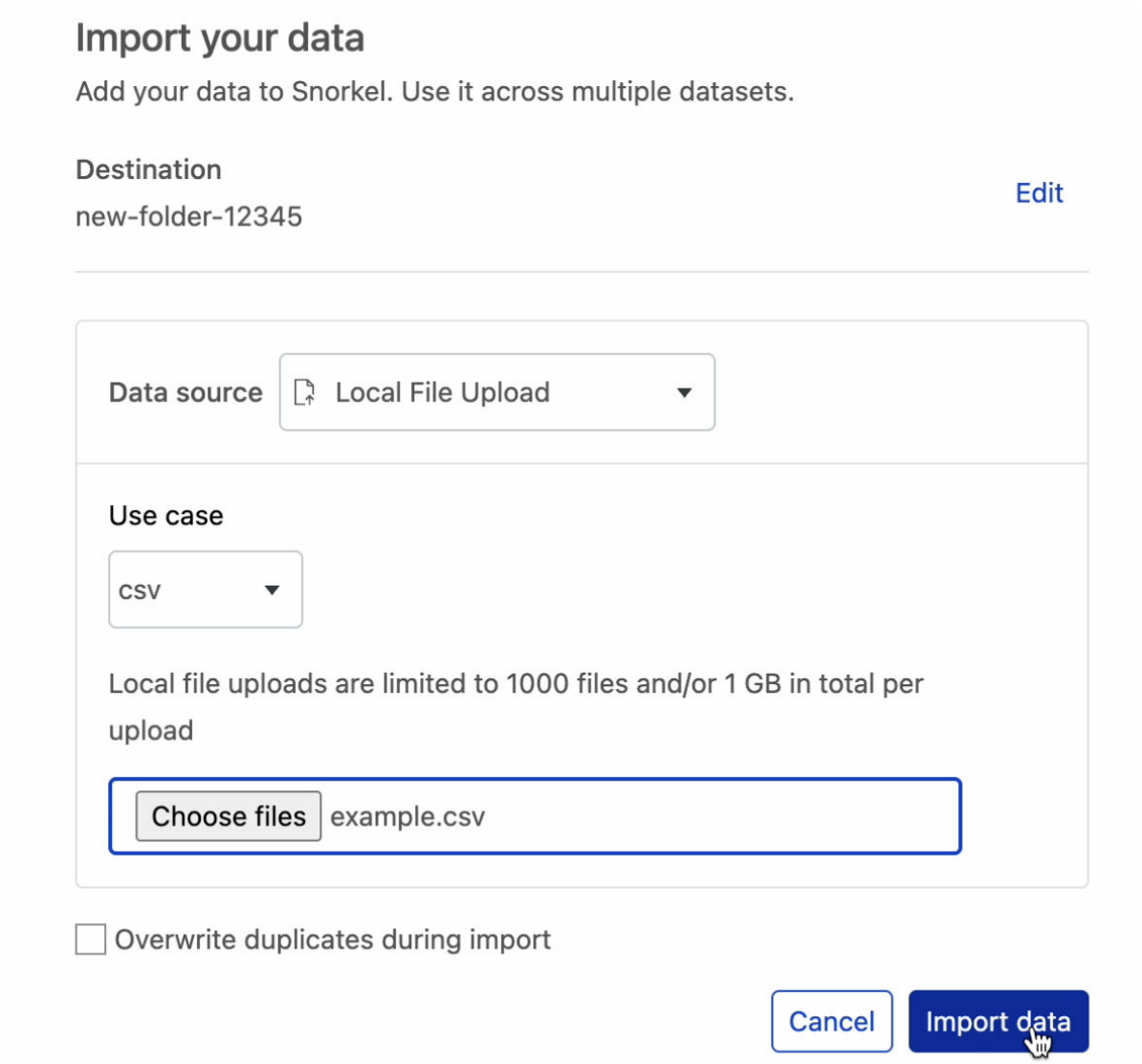
4. Choose your data source. The available sources are Amazon S3, Google Cloud Storage, and Local File Upload.



5. Select the file type from the use case dropdown. Supported use cases include uploading images, PDFs, CSVs, parquet, and arrow files.



6. Enter your file paths or select files from local storage. Select **Import data** to finish the data upload.



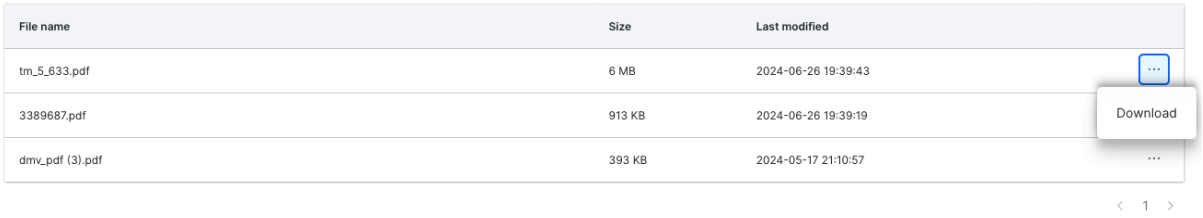
- If you select the **Remote Storage** option, your files are transferred in the background. You can check the status of this transfer using the **Jobs** icon. - If you select the **Local Storage** option, your files are uploaded. A loading spinner appears, and the user interface is disabled.

> **IMPORTANT**
> Do not close the tab. Wait for an alert to confirm the upload finished successfully. It may take several minutes to finish uploading.

Once you have uploaded or transferred files to your Snorkel AI Data Development Platform, you can associate them with your data sources when uploading new datasets.

To download files from a folder

1. In the left-hand sidebar, select the file folder from which you want to download.
2. Select the `...` icon next to an individual file to open the file options menu.
3. Select **Download**.



The file downloads to your browser's default download location.

Working with MinIO

> ⚠ WARNING
>
> **Notice of deprecation**
>
> MinIO is scheduled for deprecation in future versions of the Snorkel AI Data Development Platform through early 2025, with its functionality being replaced by **Snorkel File Service (SFS)**.
>
> - We will continue to proactively communicate changes regarding deprecation directly with customers and update documentation when appropriate. If you have concerns regarding current usage of MinIO and whether it will be supported by SFS, communicate with your Snorkel Success Manager.
> - The MinIO Console will be deprecated as of Q4 2024, but will still be accessible until the end of Q1 2025 to support backwards compatibility for older workflows. After Q1 2025, access to MinIO Console will be removed.
> - Instead, use the Data Storage feature for uploading PDFs and images. Support for arbitrary file type upload and basic file management utilities within the Files feature will be provided by end of Q1 2025 to meet MinIO Console core feature parity.
> - For files that are not PDFs and images, you can continue to use the Snorkel AI Data Development Platform's SDK. These SDK workflows are detailed below.
> - All SDK workflows documented here will be supported by SFS.

Overview

MinIO is an object store that is compatible with S3. A MinIO API is shipped with the Snorkel AI Data Development Platform for data management across the platform.

It allows users to upload and download files that are accessible from both Notebooks within the Snorkel AI Data Development Platform as well as in Operators.

Connecting to MinIO

Authentication to the Snorkel MinIO is handled automatically if you are working in Snorkel Notebooks. If you're accessing MinIO elsewhere, you will need to set the following environment variables `MINIO_URL`, `MINIO_ACCESS_KEY`, and `MINIO_SECRET_KEY`.

Using MinIO with the Snorkel SDK

When working with the Snorkel SDK, you first need to set the appropriate context to ensure code is being applied to the right workspace and application. To use MinIO, you need to define the workspace:

```
import snorkelai.sdk.client as sai

# Set your workspace
workspace_name = 'YOUR_WORKSPACE_NAME' #INPUT - Replace with your workspace
name

# Configure client context for Snorkel instance
ctx = sai.SnorkelSDKContext.from_endpoint_url(
    workspace_name=workspace_name,
)
```

This `sai` client object will be used in sections below

File upload

The SDK provides two explicit methods for uploading files (`upload_file`) and directories (`upload_dir`) using the `sf` client object defined at the top of this section. Both absolute and relative paths are supported. Examples are shown below:

```
# Upload a file from a local directory to MinIO
local_file_path = "/path/to/local/report.pdf"
remote_file_path = "minio://bucket/path/to/some/report.pdf"

uploaded_file_path = sai.upload_file(local_file_path, remote_file_path)
```

```
# Upload a directory from a local directory to MinIO
local_directory = "/path/to/local/directory"
remote_directory = "minio://bucket/upload/directory"

uploaded_dir_path = sai.upload_dir(local_directory, remote_directory)
```

File download

The SDK has two explicit methods for downloading files (`download_file`) and directories (`download_dir`) using the `sf` client object defined at the top of this section. Both absolute and relative paths are supported. Examples are shown below:

```
# Download a file from MinIO to a local directory
remote_file_path = "minio://bucket/path/to/some/report.pdf"
local_file_path = "/path/to/local/report.pdf"

sai.download_file(remote_file_path, local_file_path)
```

```
# Download a directory from MinIO to a local directory
remote_directory = "minio://bucket/upload/directory"
local_directory = "/path/to/local/directory"

sai.download_dir(remote_directory, local_directory)
```

List directory

To list files in a remote directory, use the `list_dir` method with the `sf` client object defined at the top of this section. Example is shown below:

```
remote_directory = "minio://bucket/upload/directory"

sai.list_dir(remote_directory)
```

Generic file operations

For file operations, use the SDK function `snorkelflow.utils.file.open_file`, which will return a file-like object. `open_file` works with both MinIO and local paths.

Reading a file from MinIO:

```
from snorkelflow.utils.file import open_file
with open_file("minio://bucket/path/to/some/file", mode="r") as f:
    data = f.read()
```

Writing a file to MinIO:

```
from snorkelflow.utils.file import open_file
with open_file("minio://bucket/path/to/some/file", mode="w") as f:
    data = f.write("Hello, World!")
```

Tips for splitting and partitioning data

When working with a new dataset, one of the most important steps is to create three representative splits of data.

- `train`: for LF development and training set creation
- `valid`: for model hyperparameter tuning
- `test`: for final evaluation

Given the nature of the task and the particularities of the data, it may not be appropriate to randomly split the data across these partitions.

General tips

- **Split size:** While split size is problem-dependent, a good starting breakdown is around 70/15/15 (`train`/`valid`/`test`).
- **Test split**: Make sure your `test` split contains the most reliable ground truth labels compared to other splits.
- **Hierarchical Data**: Create splits based on the hierarchical nature of your data. For example, many text extraction tasks are done at the "page level" but each page may correspond to a given document. In this case, your should split by a unique document identifier, and all of its individual pages in a given split.
- **Diverse data**: If the diversity of data is high, ensure that each split, to the degree possible, contains a representative "slice" of data from each source of diversity (i.e. stratified sampling). For example, if you want to extract fields from PDF reports, and you have noticed a dozen or so different report styles that account for 80% of the data. Ensure that each split contains some degree of each report style across `train`, `valid`, and `test`. While this is important for ensuring models can generalize, it's more critical to account for hierarchical sources of data leakage.

Snorkel specific tips

These tips are relevant when uploading a dataset.

- **Dev split:** The Snorkel AI Data Development Platform automatically generates a `dev` split from the `train` split. The default size is 10% of the total `train` size and has a hard limit of 10k examples (or 4k when sampling by documents for extraction tasks). When resampling a `dev` split close to the total dataset size, the system always reserves at least one sample for training. Because Snorkel does not support manually selecting your `dev` split, think carefully about how this could potentially cause data leakage issues. For example, if data from multiple documents are split by pages and `dev` is a random sample of pages from `train`.
- **Ensure each datasource file is < 100 MB:** If you have one large file, please repartition the file into several smaller files and ensure each file is **<100 MB** in size to reduce overhead.
- **Discrete Splits for Slices**: Consider creating multiple files for `valid` and `test` splits for specific "slices" of data, depending on the nature of your data or application. The Snorkel AI Data Development Platform supports uploading and managing multiple data sources per split, which is useful for quickly assessing model performance on a given segment or slice of data. What you choose to include in these slices will depend on the specific nature of each application.

Using data slices

A slice is a filtered subset of data rows that share a specific characteristic, like a topic, language, or error type. By using filters, users can create slices to focus on high-impact data areas for analysis, debugging, and targeted model training.

Slices are created by directly assigning individual datapoints to a slice.

> NOTE
>
> Slices have some limitations:
> - You cannot create batches from a slice.
> - Slices cannot be directly used for model training.
> - Slice definitions cannot be exported between projects.

When to use slices

There are two primary use cases for using slices within the Snorkel AI Data Development Platform—annotator and data scientist.

Annotator

As an annotator, you can create slices to focus on specific data segments that need careful labeling or review, which helps prioritize data that aligns with specific project goals.

- For more information about how to annotate, see Walkthrough for annotators.
- For more information about how to review annotations, see Walkthrough for reviewers.

Data scientist

As a data scientist, you can use slices to analyze model performance on specific subsets and improve model accuracy in targeted areas. In the **Evaluate** page, data scientists can track model performance across slices, helping identify where the model is excelling or needs improvement. In **Studio**, data scientists can filter data by slice to focus on high impact subsets for training and error analysis. Continue reading to learn more about how to create and use slices through the UI and SDK.

Where to find slices

- Annotation Suite
- Studio Page
- Evaluate Page
- Batches
- Slices Page

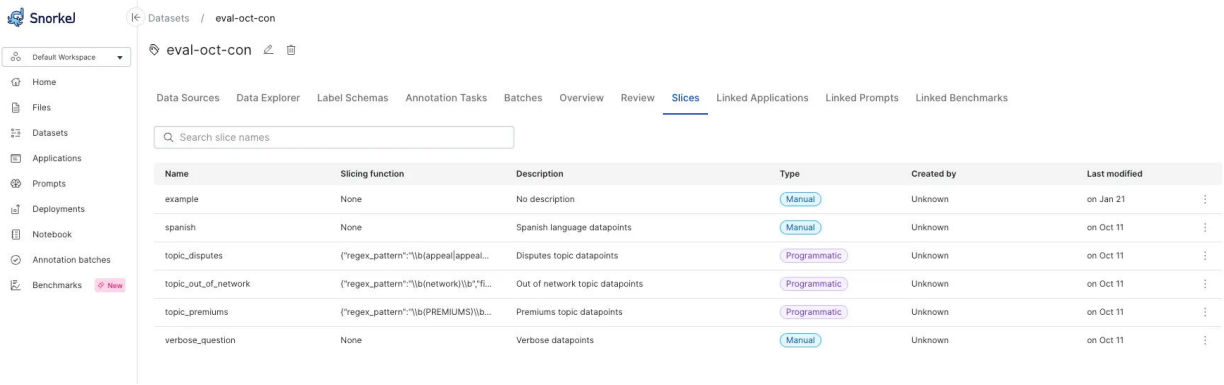
Slices page

The **Slices** page provides a centralized view of all slices in your dataset and allows you to manage them efficiently.

From your dataset, select the **Slices** tab.

On the **Slices** page, you can:

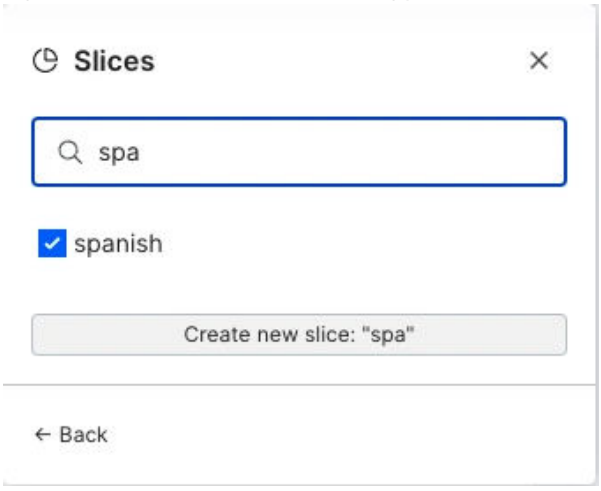
- View all slices created for the dataset.
- Search for specific slices using the search bar.
- View slice details including creation information and last modified date.
- Edit slice names and descriptions by hovering over the cell and selecting the rename icon.
- Delete slices by hovering over the slice and selecting the trash can icon.



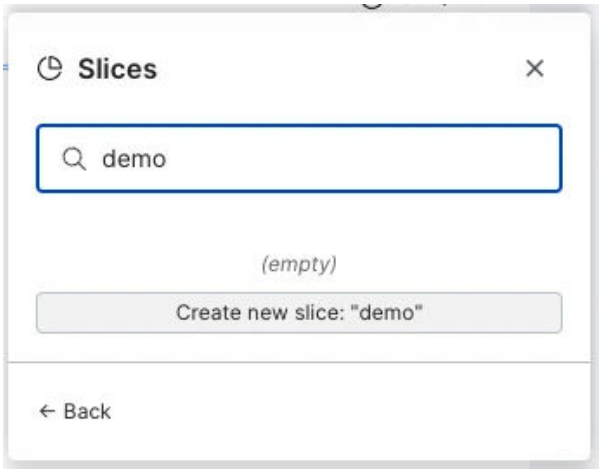
Edit slice membership

You can search for a slice, add a new slice, or check and uncheck current slices for the specific datapoint.

1. Select the **Slices** icon.
2. Select the **Edit slice membership** button.
3. Edit the slice membership:
   - If you want to search for a slice, type the slice name in the **Search or add slice** bar:



   - If you want to add a new slice, type the slice name in the **Search or add slice** bar and select **Create new slice**:

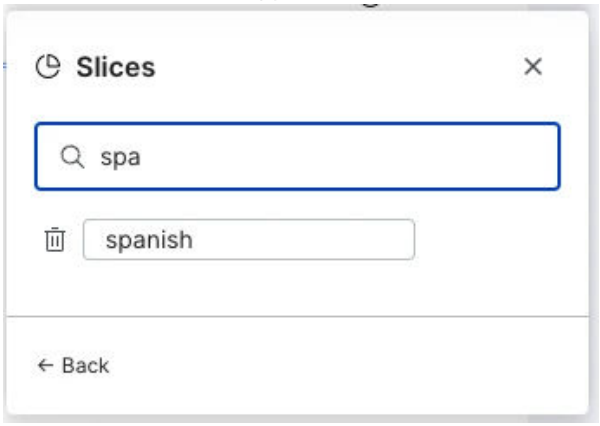


- If you want to check or uncheck current slices for the specific datapoint, select the checkbox next to the slice name.

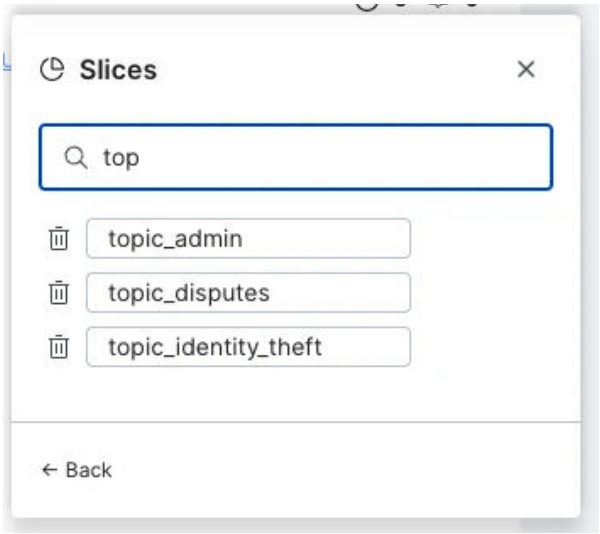
Manage slices

You can search for a slice, delete a slice, or edit the name of a slice.

1. Select the **Slices** icon.
2. Select the **Manage slices** button.
3. Manage the slice:
   - To search for a slice, type the slice name in the **Search** bar:



   - To delete a slice, select the **Delete** icon to the left of to the slice name:



- To edit the name of a slice, type the updated name in the specific slice bar.

Manage slices using the SDK

In the SDK, use the `slice` class to create and manage subsets of data slices within a dataset. Slices are manually defined and accessed through the `create()` and `get()` methods.

1. Initialize a slice. Use `Slice.create()` to define a new slice by specifying the dataset, name, and optional description.
2. Add or remove datapoints. Use `add_x_uids(x_uids)` to add specific datapoints by UID to a slice.
3. Retrieve and list slices:
   - Use `Slice.get(dataset, slice)` to retrieve a specific slice.
   - Use `Slice.list(dataset)` to list all slices associated with a dataset.
4. Updating and managing slices:
   - Modify properties like name or description using `update([name, description])`.
   - Remove datapoints by UID from a slice using `remove_x_uids(x_uids)`.

Annotation Studio overview

Data **annotation** is the process of assigning labels or classes to specific data points for training datasets. For example, you could use an LLM to classify banking contract documents to one of the following classes:

- employment
- loan
- services
- stock

You would collect annotations to define the **Ground truth (GT)** to evalaute the LLM's classification performance.

Ground truth (GT) refers to the set of labeled and accurately annotated data that serves as a reference or benchmark for training and evaluating machine learning models. During the evaluation phase, use ground truth data to assess the model's performance. By comparing the model's predictions with the actual labels in the ground truth, you can calculate metrics such as accuracy, precision, recall, and F1 score.

Annotation Studio is divided into four sections:

- **Overview page:** Provides aggregate metrics on the number of completed annotations, the distribution of labels annotator agreement, and a view into recent annotator activity. See Overview page: View aggregate metrics for more information.
- **Review page:** Provides a full list of all annotations and their corresponding documents to easily review all annotations in one place. See Walkthrough for reviewers for more information.
- **Batches page:** Provides access to and information about all batches that have been created. You can create and manage batches and commit annotations to ground truth. See Create batches and Manage batches and commit ground truth for more information.
- **Within a batch:** Provides a canvas where annotators can view and label documents. See Walkthrough for annotators for more information about how to annotate in Annotation Studio.

The Snorkel AI Data Development Platform provides multiple user roles that administrators can assign, each with different levels of data access and permissions. See Roles and user permissions for more information.

Why do we need to do manual annotation in Snorkel?

While the Snorkel AI Data Development Platform programmatically generates labels for your training data sets, you need to define your initial ground truth to develop labeling functions (LFs) and models. In addition, manual annotation allows for an iterative process of model development. Annotators can review model predictions, identify errors, and refine the labeled dataset. These manual annotations lead to improved model performance. With Snorkel, you can pinpoint where manual annotation is needed, which significantly reduces the time and money spent on manual annotation.

Typically, you'll want multiple annotators to review each data point for these reasons:

- **Reduce bias:** Manually labeling data is prone to bias because the person labeling the data may have preconceptions that can influence the labels they assign.

- **Handle ambiguity:** Some documents might be inherently ambiguous or open to interpretation. In these cases, having multiple annotators allows for capturing different viewpoints and addressing the inherent uncertainty in the data.
- **Enhance robustness:** By aggregating annotations from multiple annotators, the labeled dataset becomes more robust and less dependent on the idiosyncrasies of any single annotator. This robustness is particularly important when dealing with diverse datasets or complex tasks.
- **Refine annotation guidelines:** Comparing annotations from multiple annotators identifies areas where guidelines need clarification. It also provides an opportunity for continuous training to improve consistency and understanding among annotators.

Use keyboard shortcuts

These shortcuts are configured for dataset-level annotation views. Also, see annotation settings.

Record view

Shortcut	Description
Right Arrow (→)	Navigate to next data point
Left Arrow (←)	Navigate to previous data point
Up Arrow (↑)	Navigate to previous candidate, only available in extraction tasks
Down Arrow (↓)	Navigate to next candidate, only available in extraction tasks

Document view

Shortcut	Description
Right Arrow (→)	Navigate to next data point
Left Arrow (←)	Navigate to previous data point
Up Arrow (↑)	Navigate to previous candidate
Down Arrow (↓)	Navigate to next candidate

Snippet view

Shortcut	Description
Right Arrow (→)	Navigate to next page of data
Left Arrow (←)	Navigate to previous page of data

Table view

Shortcut	Description
Right Arrow (→)	Navigate to next page of data
Left Arrow (←)	Navigate to previous page of data

Guideline goal

Annotation guidelines allow users to describe a phenomenon or concept as generally and precisely as possible. Good annotation guidelines are helpful for subject matter experts to annotate the concept at hand in any text without running into problems or ambiguity issues.

An iterative process

Developing annotation guidelines is an iterative process. Once a pilot annotation, or first draft, is created, its shortcomings still need to be identified and fixed. For instance, issues such as unclear label definitions can be identified in the first draft, and fixed for the second draft. The second draft is also looked over for further shortcomings, which repeats the revision process and leads to the most polished version of the pilot annotation.

Note

The most important idea about the iterative nature of the annotation process is that in each round, **multiple annotators independently annotate the same text**.

Pilot annotation

The first round of annotations is best done by annotators who are familiar with the data and context around the data used. As with subsequent annotation rounds, please annotate in parallel and discuss afterwards. It's not necessary to spend a lot of time on preparation. Specifying a list of references or theoretical works, or agreeing on a single text should be sufficient as a starting point.

Your time is spent best on discussing annotation disagreements. In particular in the very first round, many parameters are still undecided and likely to cause disagreement.

Walkthrough for annotators

This page walks through the process of manually annotating documents in Annotation Studio. This walkthrough is designed for users with the Annotator role, who need to access the Snorkel AI Data Development Platform to annotate the documents that are assigned to them. Below, you'll find an interactive demo showing how to annotate your data.

Visit the Snorkel docs site to interact with this element

The instructions below further outline what you saw in the interactive demo.

Access your batches

After you log in to the Snorkel AI Data Development Platform, you need to access your batches to start annotating. Batches are a collection of documents that are assigned to you to manually annotate. A batch is typically a subset of the total documents. With the Snorkel AI Data Development Platform, the number of documents that need to be manually annotated is significantly smaller.

To access your batches, click **My work** on the left-side menu.

The **My work** page shows a list of batches for you to annotate:

Batch	Status	Annotators	Annotated	Size	Created
batch-2-1-3	In progress	ronnie.reviewer, alex.a...	593	978	01/23/2024
batch-2-1-4	In progress	ronnie.reviewer, alex.a...	597	820	01/23/2024
batch-2-1-2	In progress	ronnie.reviewer, alex.a...	592	815	01/23/2024
batch-1-1	In progress	ronnie.reviewer, alex.a...	1463	1952	01/23/2024

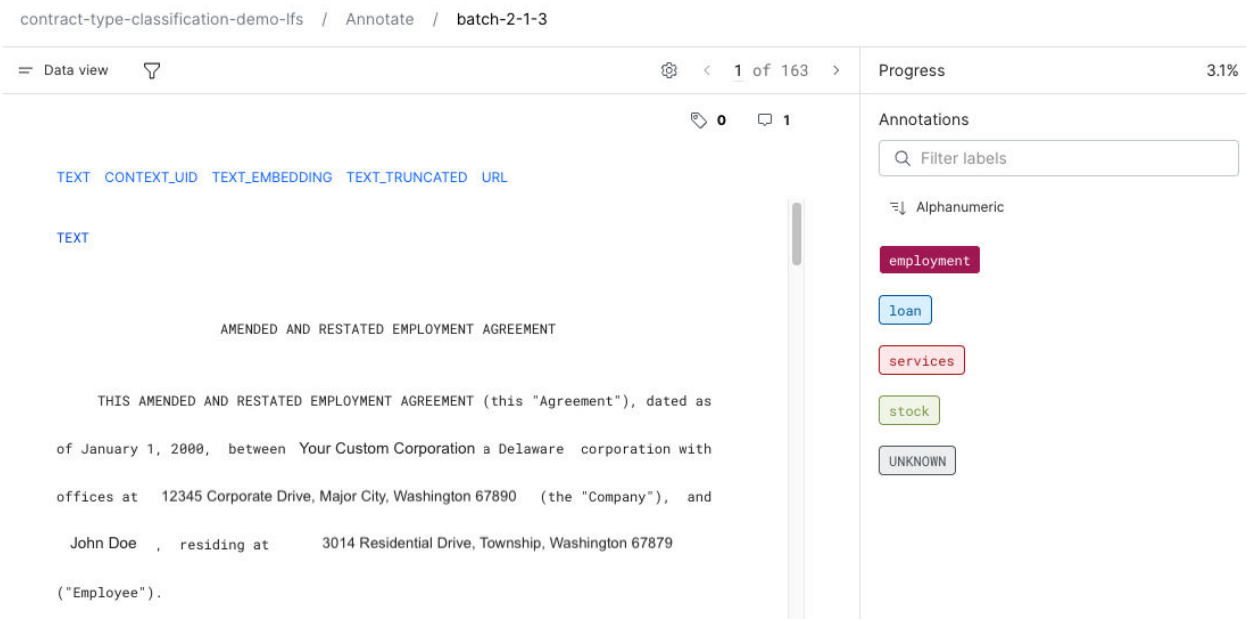
The following information about each batch is available:

- **Batch**: The name of the batch.
- **Status**: Specifies whether annotations on the batch have not be started, are in progress, or have been completed.
- **Annotators**: The list of annotators that have been assigned to annotate the batch.
- **Annotated**: The number of documents in the batch that have been annotated.
- **Size**: The total number of documents that are in the batch.
- **Created**: The date that the batch was created.

Now, you can start annotating your documents!

Annotate your documents

Once you click **Annotate**, your screen will look similar to the image below:



The main canvas shows the document text. For classification and extraction applications, the right side-pane shows all classes that you can label the document.

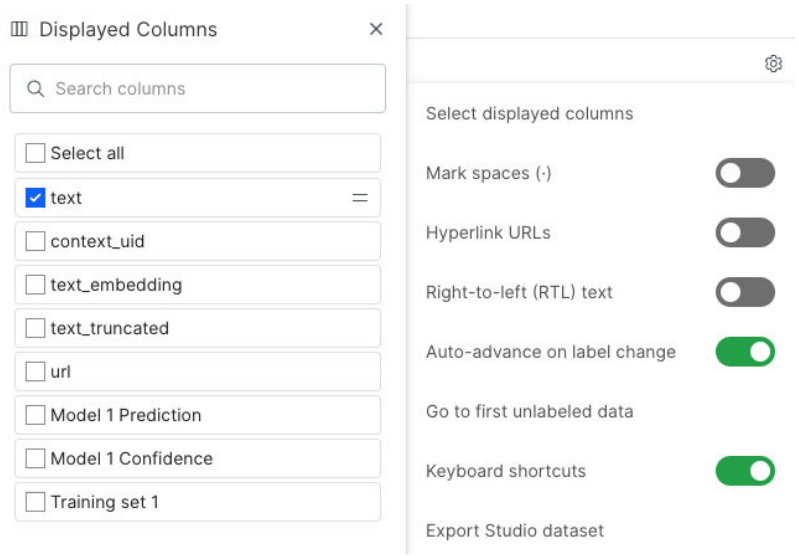
Here is some information about the various buttons that you can see on your screen:

- Click **Data view** (or **Document view** for extraction applications) to change the appearance of the documents in the main canvas. The views that are available differ based on the application type.
- Click the filter icon to filter the documents that are shown to you.
- Click the gear icon to adjust various settings. We recommend that you adjust a few settings before you get started annotating.
- Click the arrow icons to page through the documents.
- Click the slice icon to add slices to the document.
- Click the comment icon to add comments to the document. For example, you can write a comment to explain your reasoning behind a selecting a particular label.

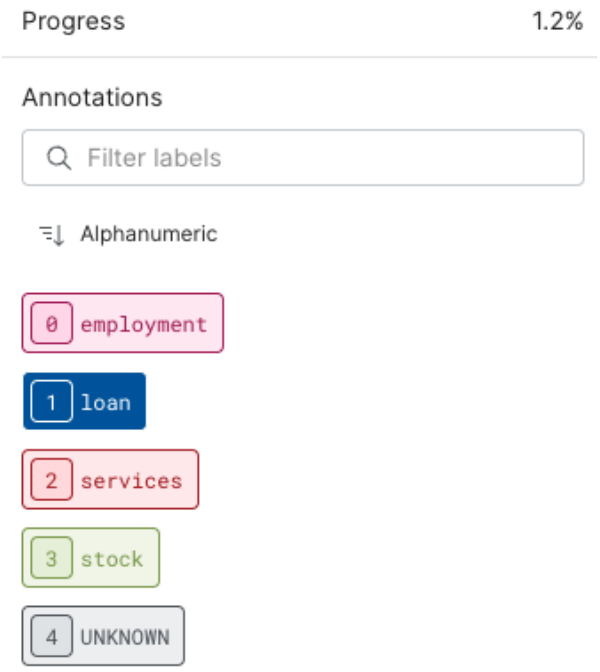
Adjust settings

After accessing your documents, we recommend that you first adjust some settings to get some quality of life improvements while annotating! To change your settings, click the **gear** icon at the top-right corner of your screen. We recommend changing the following settings:

- **Select display columns**: This setting lets you select and display only the data columns that are necessary for marking your annotations (e.g., the text column). This removes clutter from the main canvas when you are reviewing documents.



- **Auto-advance on label change**: With this setting enabled, after you label a document, you automatically go to the next document. This prevents you from having to manually click the **arrow** > icons to page through the documents every time to view new, unlabeled documents. This setting is only available for single-label and text extraction applications.
- **Keyboard shortcuts**: With this setting enabled, keys are assigned to each label class, which makes it quicker to assign a label. In the example below, if you want to label a document "employment," then you just have to press 0 on your keyboard.



Here are descriptions of the rest of the settings that are available when you click the **gear** icon:

- **Mark spaces (·)**: Replaces all spaces in your document with a dot (·).
- **Hyperlink URLs**: Hyperlinks any URL in your document. This makes it easier to see links in your documents and allows you to click the hyperlink to go directly to the website.
- **Right-to-left (RTL) text**: Right justifies your document text. By default, the document text is left justified.

- **Go to first unlabeled data**: Brings you to the first document in the batch that has not yet been annotated. This option is helpful any time you re-enter a batch to continue annotating.
- **Export Studio dataset**: Export your dataset into a CSV file.

How to Annotate

The following sections walk through how to annotate documents for the four different task types:

- single-label classification
- multi-label classification
- text extraction
- sequence tagging

Single-label classification applications

In single-label classification applications, your goal is to assign a single class to each document. For example, assigning banking contract documents to one of the following classes: "employment," "loan," "services," or "stock." For more information about this setting, see Set the default label for multi-label annotation applications.

On the right-side pane, you'll see all possible classes that you can label your document. To label a document, click the class in the right-side menu. Then click the arrow button to move on to the next document. If you adjusted the recommended settings, then you use can use the keyboard shortcut to assign a class, and then you will automatically be brought to the next document.

Multi-label classification applications

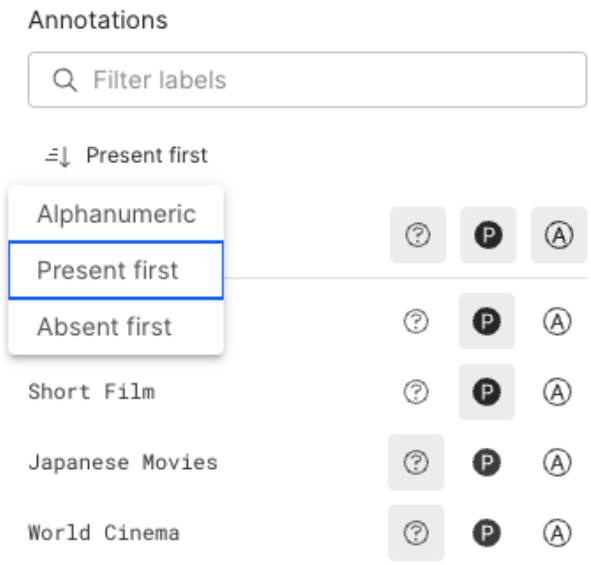
In multi-label classification applications, individual documents can have multiple label values. For example, let's say you are looking at movie review documents. You can label the movie as "Short Film," "Black and White," "Japanese Movies," or "World Cinema." Given these labels, you can see that a single movie can fall into multiple categories. In this case, for a given document, you can label each possible class as present, absent, or abstain from voting.

On the right-side pane, you'll see all possible classes. For each class, you can click:

- ? to label the class as Abstain.
- P to label the class as Present.
- A to label the class as Absent.

By default, all classes are initially labeled as Abstain. If enabled for your application, you can set the default label for each class to either Present, Absent, or Abstain.

You can also sort the classes, which is particularly helpful for applications with a large number of classes. You can sort classes marked as **Present first**, classes marked as **Absent first**, or in **Alphanumerical** order.



(Beta) Set the default label for multi-label annotation applications

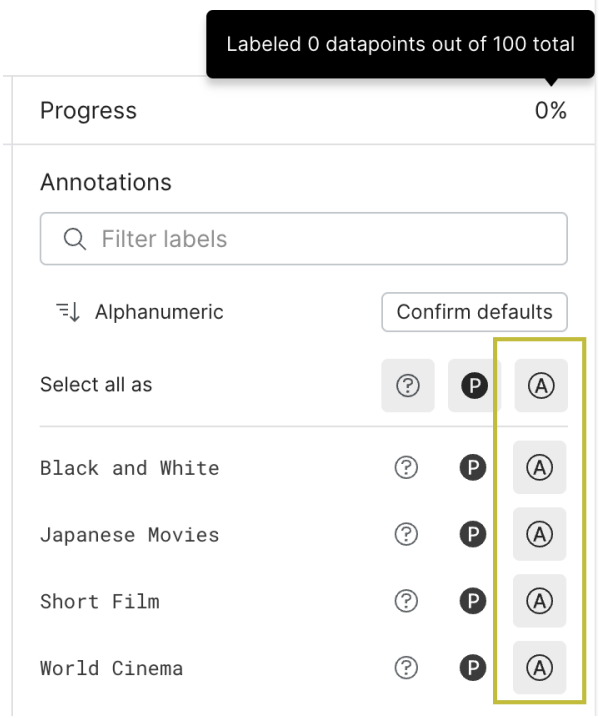
> NOTE
> This is a beta feature available to customers using a Snorkel-hosted instance of the Snorkel AI Data Development Platform. Beta features may have known gaps or bugs, but are functional workflows and eligible for Snorkel Support. To access beta features, contact Snorkel Support to enable the feature flag for your Snorkel-hosted instance.

In multi-label classification applications, individual documents can have multiple label values. For a given document, you can label each possible class as present, absent, or abstain from voting. From the **Overview** page, you have the option to set the default label value for your application. This is helpful in situations where different teams in your organization want their default label to be different based on their particular use case. In addition, this setting is application specific so it will only affect your current working application.

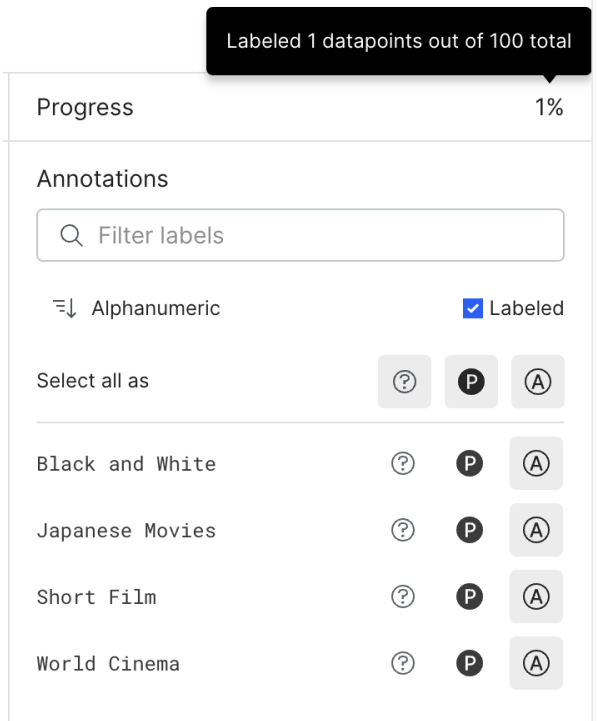
> NOTE
> This functionality is a beta launch that is only made available on a request basis. Please contact your Snorkel representative if you are interested in getting access.

Follow the steps below to set up your default label:

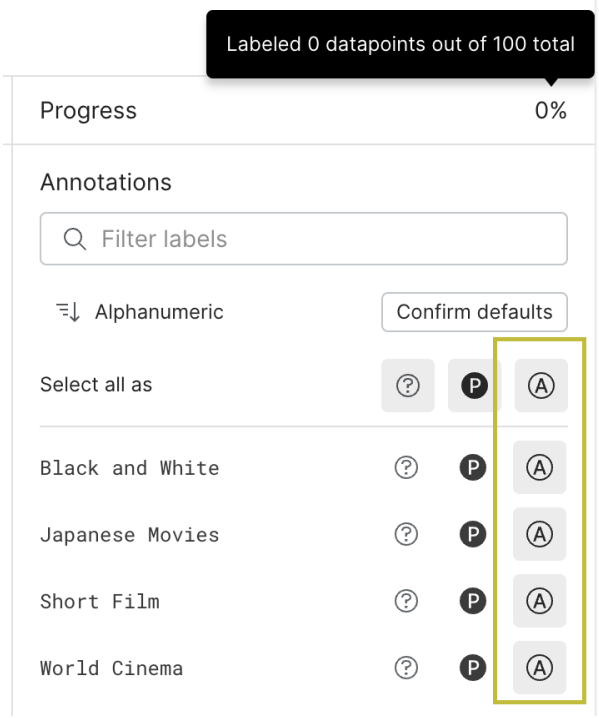
1. Click the menu in the top right corner of your screen.
2. In the modal, select a default label of **Present**, **Absent**, or **Abstain** from the dropdown.
3. Click the **Save** button to save your changes.
4. Open a batch and then you'll see that for unlabeled documents, the default label is set as your selection.



5. You can now take one of the following actions when reviewing a document:
   a. You can select labels however you like.
   b. You can go with the default labels by clicking the **Confirm defaults** button.
   You can see the progress has gone up after you've completed either of the actions listed above and the state has changed to **Labeled**.



6. If you want to revert your changes and go back to the unlabeled state, click **Labeled**. The modal asks for your confirmation as this action cannot be undone.
7. After you confirm, the document returns to an unlabeled state. You'll also see the **Confirm defaults** button again along with the default labels and progress.



TIP

You can also change your default label while annotating your batches. This action will not affect any of the labeled documents and only reflect the default label for unlabeled ones.

Text extraction applications

In text extraction applications, the goal is to extract key information from a document. For example, let's say that we want to extract all dates from a document. In this case, a date is considered a span. Your goal in this example is to review all highlighted spans in a document and identify whether the highlighted spans are in fact dates. You can label each highlighted span as "NEGATIVE," "POSITIVE," or "UNKNOWN" in the right-side pane. In this example, "POSITIVE" means that the highlighted span is a date, and "NEGATIVE" means that the highlighted span is not a date.

If you adjusted the recommended settings, then you can use the keyboard shortcuts to label a span, and then automatically to be brought to the next highlighted span in the document.

By default, you will be viewing the spans in Document view. We recommend annotating in either Document view or Span view:

- **Document view** shows the entire document on the main canvas. You can use the up and down arrows on your keyboard to navigate between spans of text to annotate. This view provides you with all context surrounding a span.
- **Span view** shows a single span per page. With this view, you don't have all the context surrounding a span. However, the simplified view can enable you to more quickly annotate things like dates that are easy to identify with less context.

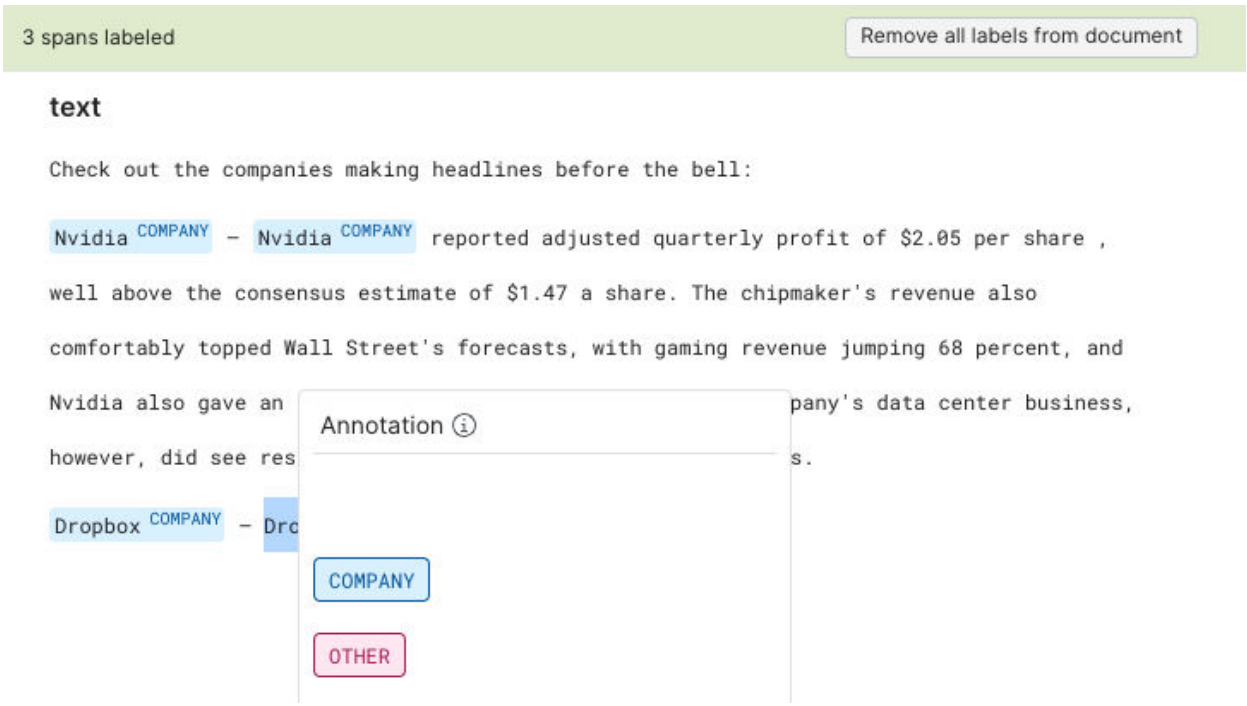
Sequence tagging applications

In sequence tagging applications, your goal is to highlight and label spans throughout a document. Spans are key pieces of information that you want to extract from a document. For example, let's say we want

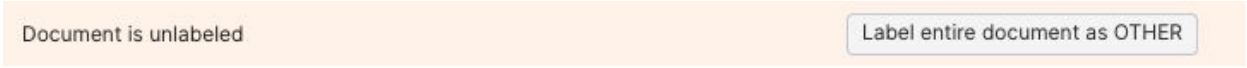
to identify all mentions of company names in a document. In this case, a company name is considered a span. Your goal is then to highlight and label all company names that you find while reading through the document.

To label spans in Annotation Studio:

1. Highlight a span.
2. Select the label that you want for that span in the Annotation modal.



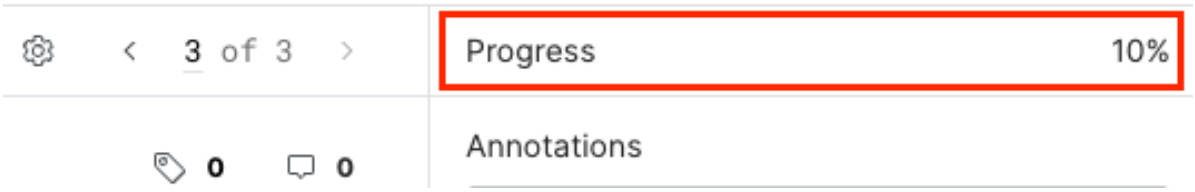
If you don't find any spans to label in the document, you can indicate this by clicking the **Label entire document as OTHER** button above the document text.



At any point, you can reset and clear all spans that you have labeled in the document by clicking the **Remove all labels from document** button above the document text.

Check progress

While you are annotating documents, you can see your progress percentage tick up in the right-hand corner of your screen.

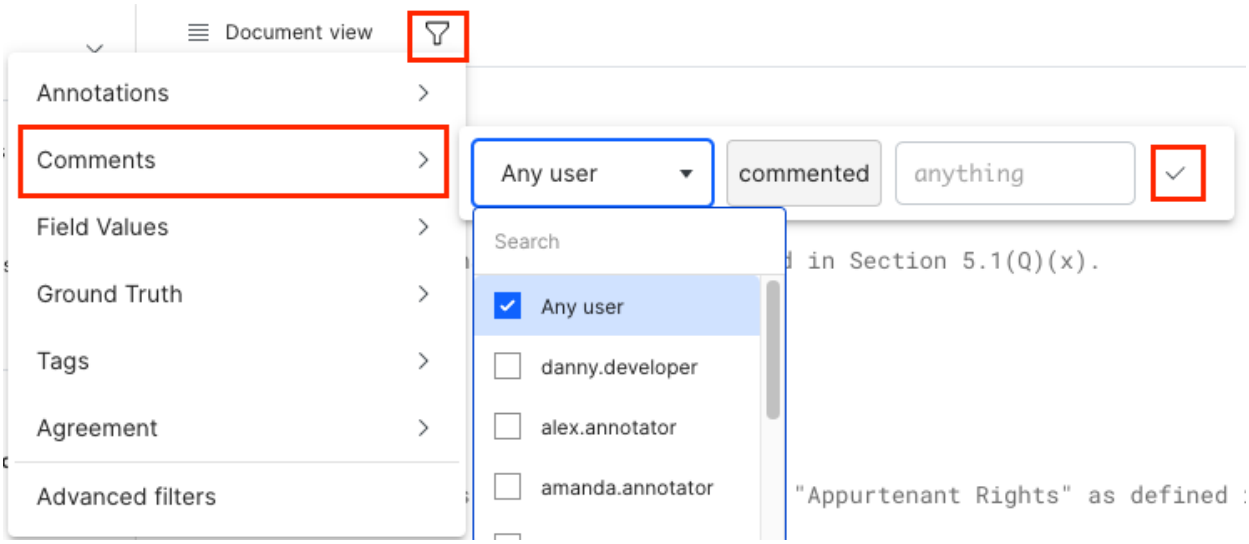


Once you are finished, your progress bar will be labeled 100%, and the status will be updated to **Complete** on the Batches page.

Review comments and slices

Occasionally a reviewer will leave comments on documents. For example, if you left a comment asking a clarifying question, a reviewer may respond to your comment. To see all documents with comments:

1. Click the filter icon.
2. Click **Comments**.
3. Under **User**, select the person that you want to see comments from. Alternatively, you can select **Any user**.
4. Click the checkmark ✓ to save the filter.



Now, just the documents with comments are shown, making it easier for you to review any comments.

You can follow similar steps to filter documents based on slices:

1. Click the filter icon.
2. Click **Slice**.
3. Under **Slice**, select a slice option.
4. Under **Operator**, select **is** if you want to see all documents with that slice or **is not** if you want all documents with that slice removed from view.
5. Click the checkmark ✓ to save the filter.

Walkthrough for reviewers

This page walks through the different ways that users with the Reviewer role can review annotations from all annotators.

As a reviewer, you will likely be tasked to manually annotate documents, in addition to reviewing annotations from others. Follow the Walkthrough for annotators to get step-by-step instructions annotating documents for the different task types.

You can review annotations in two places:

- Within a batch: View all annotations made on a single document side-by-side.
- On the Review page: View a full list of all annotations and their corresponding documents.

In addition, you have some additional permissions that allow you to manage batches.

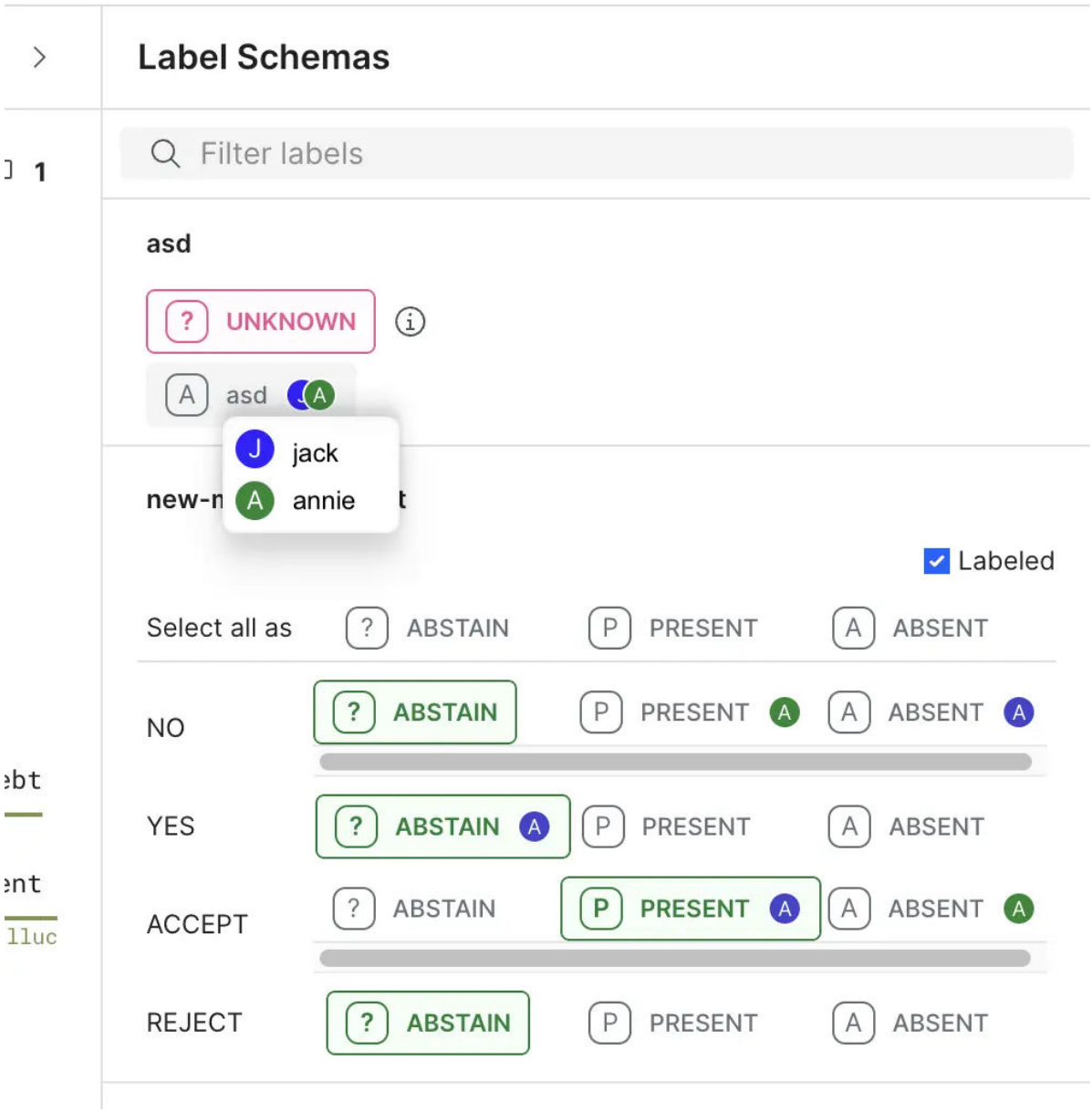
Review documents within a batch

While inside a batch that you are assigned, being a reviewer allows you to review other users' annotations. With multi-schema annotation, reviewing becomes a superset of annotating. In this view, you can see all annotations by toggling the **Reviewer** mode on and off.

How you review the annotations differs based on the task type. For all task types, you can click the comment icon to add an explanation for why you agree or disagree with other annotations. In addition, you can click the slice icon to add a slice to the document. For example, you may want to create a slice called **contains-disagreements** to slice documents that have significant disagreement among the annotators.

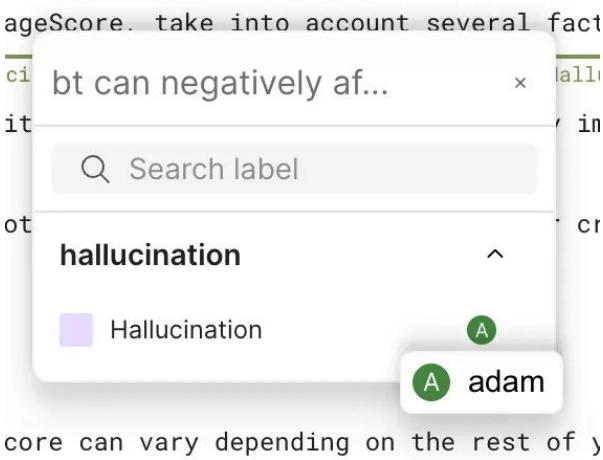
Review classification and extraction annotations

For classification and extraction applications, you can see all annotations made by other annotators. You can also see the annotator's name and the annotation label that they selected.



Review sequence tagging annotations

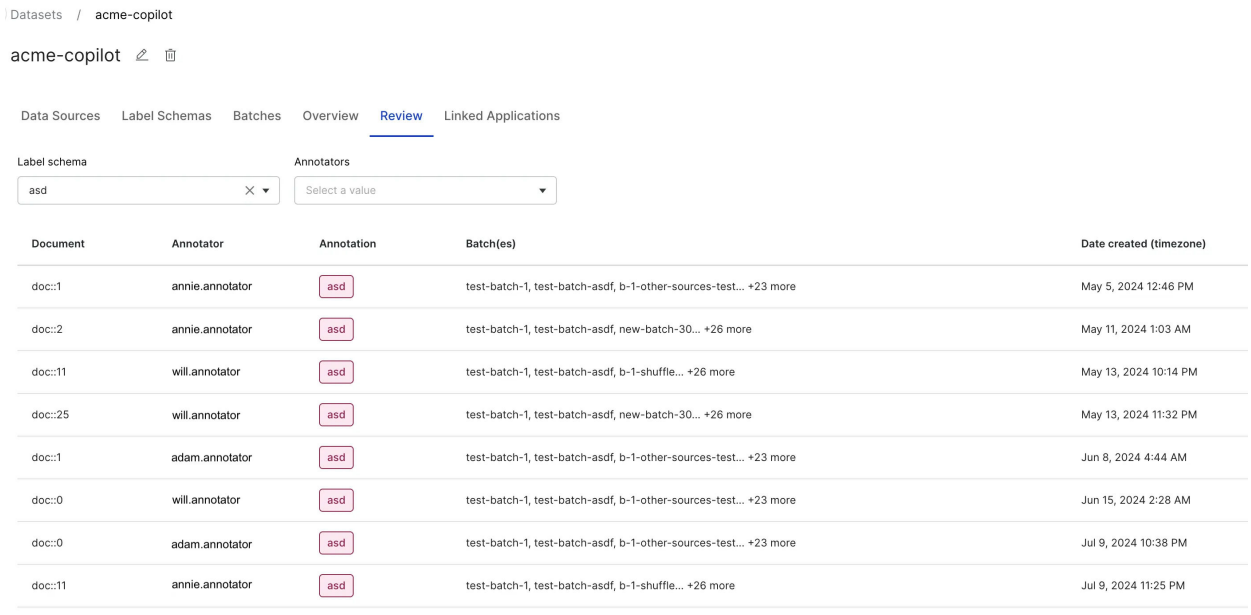
In sequence tagging applications, you can see annotations in the label pop-up when you select an annotated span.



Review all annotations on the review page

The Review page provides a full list of all annotations and their corresponding documents to easily review all annotations in one place. To access the Review page:

1. Click **Datasets** in the left-side menu.
2. Click the name of the dataset that you want to review.
3. Click the **Review** tab.



Select an annotation to pull up the full document text. Here, you have the option to leave comments and add slices to the document. In addition, you can select the filters icon on the top-right corner of your screen to filter the table by a particular annotator.

Manage batches

As a reviewer, you have additional permissions that allow you to manage batches:

- Rename and delete batches.
- Aggregate annotations in a batch.

- Set an annotator as an expert for that batch to view the agreement rate for each annotator relative to the expert.

Create batches

This page walks through how you can create batches of documents for manual annotation when the dataset uses multi-schema annotations.

> **NOTE**
>
> To create batches and commit annotations to ground truth, you must have the Developer role. For more information about access and permissions, see Roles and user permissions.

What are batches?

Before documents can be annotated in Annotation Studio, they must be assigned to batches. A batch is simply an arbitrary collection of data points—it can be as large as an entire split or data source or as small as a few examples.

Typically, you'll want documents manually annotated at a couple different points during the data development process:

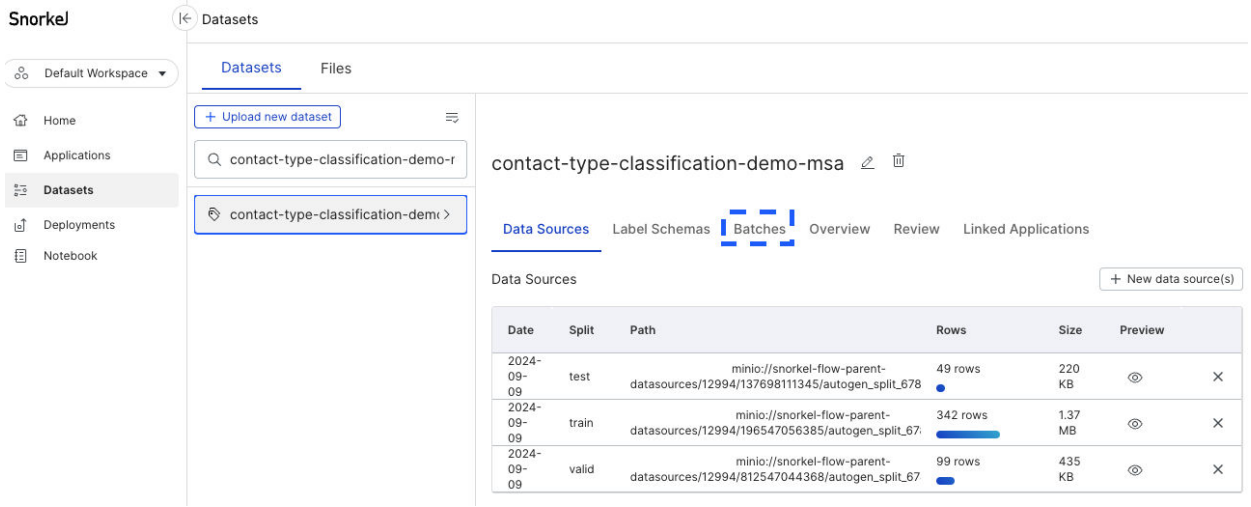
- At the beginning of a project so that you can get some initial ground truth to begin development.
- During labeling function (LF) and model development. The development cycle is an iterative process and you may want to introduce more datapoints with ground truth. The reasons may be to improve the inbalances in the dataset, or simply to add more training data.

How to create batches

There are two ways that you can create batches in the Snorkel AI Data Development Platform:

- **Dataset page:** Best method to use if you want to quickly create batches for an entire split or data source.
- **In-platform notebook:** Best method to use if you have a specific list of `x_uid`s that you want to create batches from.

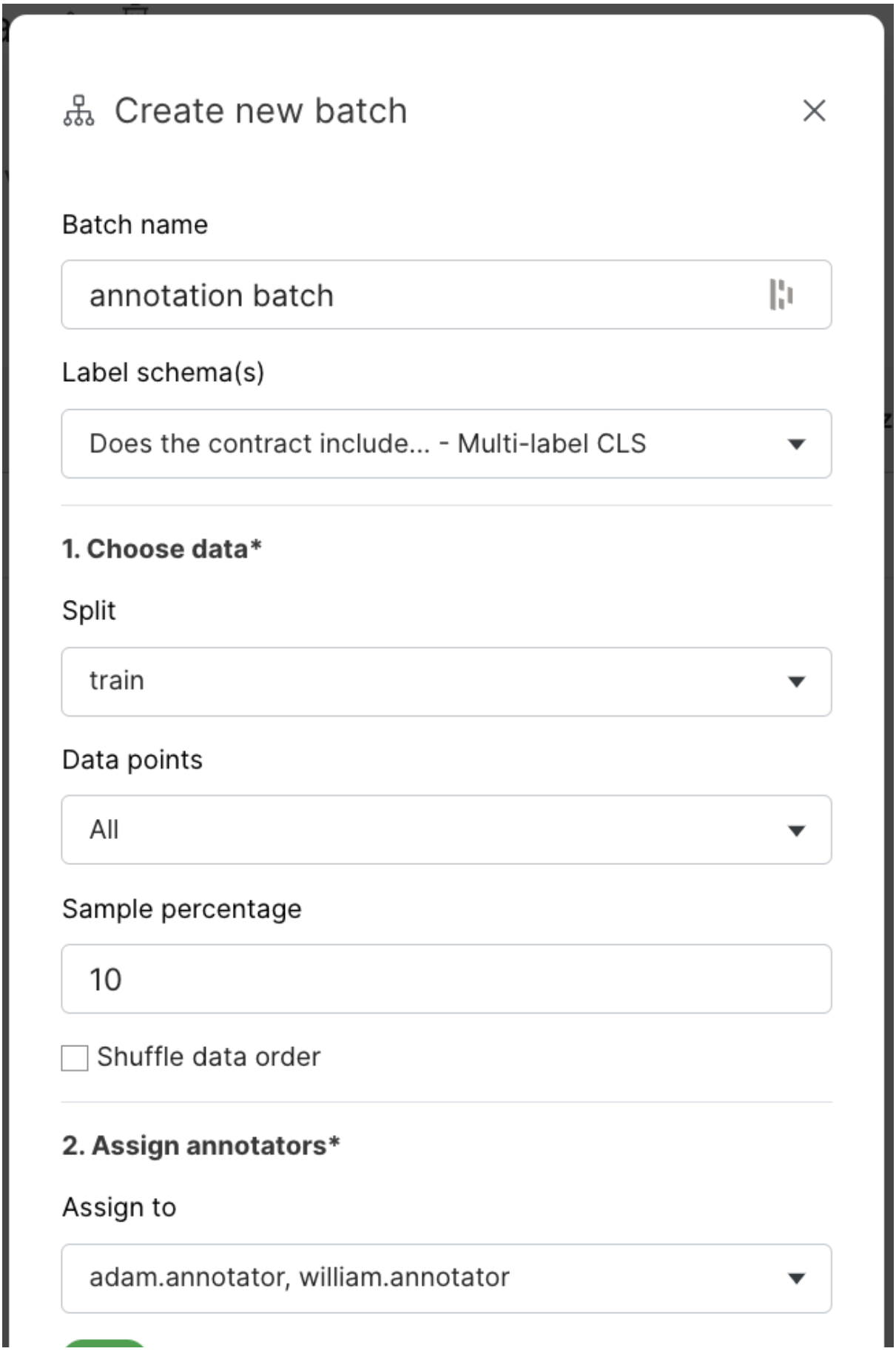
All batches that you create can be seen and managed in the **Batches** tab. This page can be accessed by going to the dataset page, then clicking **Batches** as shown below.

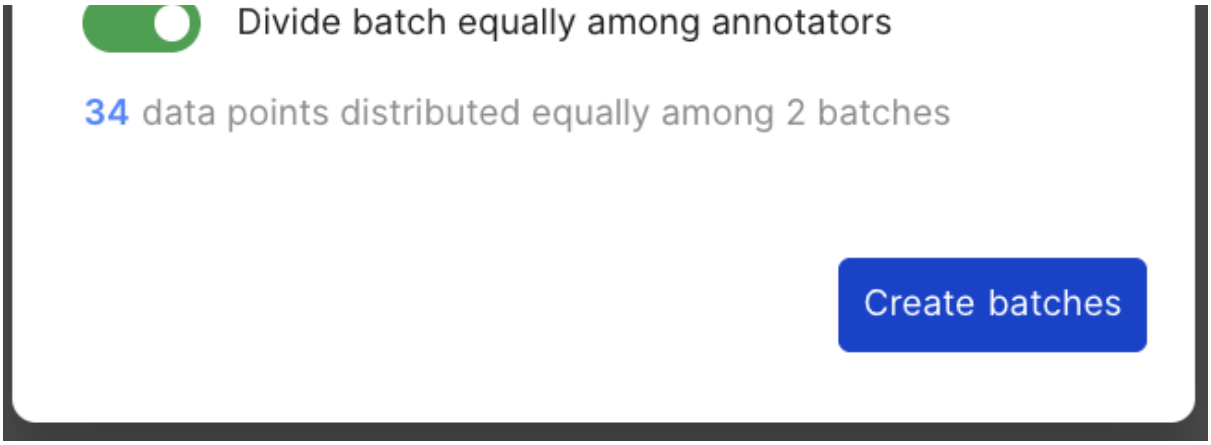


Create batches in the Dataset page

The easiest way to create batches is in the **Batches** tab from the selected dataset page.

1. Go to the **Batches** tab.
2. Select **+ Create a new batch** to bring up the **Create new batch** modal.
3. Specify the following options:
   - **Batch name:** The name for the batch. If you choose to create multiple batches by enabling **Divide batch equally among annotators**, then each batch is named the specified name with an appended numerical index to differentiate the batches.
   - **Label schema(s):** Specifies the label schema(s) to use for the batch.
   - **Select a split:** Specifies which split to use to create the batch. Each batch comes from a single source: a split, a data source, or an existing batch.
   - **Datapoints:** Specifies the data points to include in the batch. You can select all data points or data points that are not assigned to any batches.
   - **Sample percentage:** Specifies the percentage of data points to use for the batch between 1-100%.
   - **Shuffle data order:** Shuffles the order of the data points in your data source. This option enables you to randomly sample which data points go into each batch.
   - **Assign annotators:** Specifies the annotators to assign the batch to.
   - **Divide batch equally among annotators:** If enabled, datapoints are divided equally to *n* batches where *n* is the number of annotators specified in **Assign annotators**.
4. Once you finish your selections, click **Create Batch**.





The new batch or batches can now be seen on the **Batches** tab.

Create batches with the in-platform notebook

You can also create batches using the SDK in the in-platform notebook. This method is convenient if you have a specific list of `x_uid`s that you want to create batches from.

You can create batches in the SDK using the `Dataset.create_batches` function. You can search for this function in the SDK documentation for more information about all available parameters.

```python
from snorkelai.sdk.develop import Dataset

# First, get the dataset object
ds = Dataset.get(name="My Dataset")

# Example 1: Creating a single batch from all data points in the train split.
batch_list_1 = ds.create_batches(
    name="notebook_batch_from_split",
    assignees=[1, 2], # User UIDs
    split='train'
)
print(f"Created {len(batch_list_1)} batch(es).")

# Example 2: Creating a batch from a specific list of data points.
batch_list_2 = ds.create_batches(
    name="notebook_batch_specific_uids",
    assignees=[1],
    x_uids=["doc::101", "doc::102", "doc::105"]
)
print(f"Created {len(batch_list_2)} batch(es).")

# Example 3: Creating a set number of random batches from a split.
# This creates 5 randomized batches from the training split.
batch_list_3 = ds.create_batches(
    name="randomized_training_batches",
    assignees=[1, 2],
    num_batches=5,
    split='train',
    randomize=True,
    random_seed=42
)
print(f"Created {len(batch_list_3)} randomized batches.")

# Example 4: Creating batches from data points not yet in any other batch.
# This is useful for incremental labeling to ensure you are not reviewing the
same data.
batch_list_4 = ds.create_batches(
    name="new_unseen_data",
    assignees=[3],
    batch_size=50,
    split='train',
    filter_by_x_uids_not_in_batch=True
)
print(f"Created {len(batch_list_4)} batch(es) from un-batched data.")

# Example 5: Creating and dividing a batch of data evenly among assignees.
# This creates one batch per assignee from a total of 90 data points.
batch_list_5 = ds.create_batches(
```

```
    name="divided_batch",
    assignees=[1, 2, 3],
    split='train',
    batch_size=90, # Total data points to select and divide
    divide_x_uids_evenly_to_assignees=True
)
# This will create 3 batches, each with 30 data points.
print(f"Created {len(batch_list_5)} batches by dividing data evenly.")
```

The created batches can be seen and managed from the **Batches** page at the dataset page.

Annotate traces

This topic explains how you can annotate trace datasets. Once you have uploaded a trace dataset, you can begin annotating it.

1. Create a batch for the trace dataset through a Benchmark or directly from **Dataset** > **Batches**.

   > NOTE
   >
   > In the batch creation modal from **Datasets**, the data selection will show the number of traces and datapoints (i.e. total steps across all traces) for the batch.

2. Annotate against a trace dataset.
   1. Select **Annotate** for a traces batch to go to the **Trace** view.
   2. Apply each label schema at an individual step of a trace.
   3. Select a step to annotate on by searching, filtering, or directly selecting a step from the Trace navigator on the left.

   > TIP
   >
   > While global trace-level annotation isn't available, you can achieve the same goal by annotating at the Root step.

3. Commit the ground truth for a trace dataset. To do so, aggregate and commit ground truth for the batch of traces, just like any other dataset.

The committed ground truth will now appear in your associated benchmark(s) for analysis.

Manage batches and commit ground truth

This page walks through how to manage your batches and commit annotations to your ground truth to be used for development in Studio.

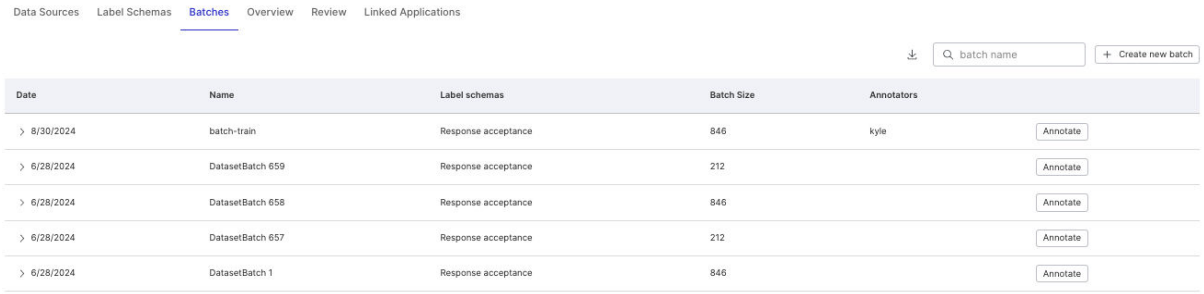
Ground truth refers to the set of labeled and accurately annotated data that serves as a reference or benchmark for training and evaluating machine learning models. During the evaluation phase, ground truth data is used to assess the model's performance. By comparing the model's predictions with the actual labels in the ground truth, metrics such as accuracy, precision, recall, and F1 score can be calculated.

The Snorkel AI Data Development Platform allows you to programmatically generate labels for your training split. Snorkel recommends using ground truth (GT) labels for data points in other splits:

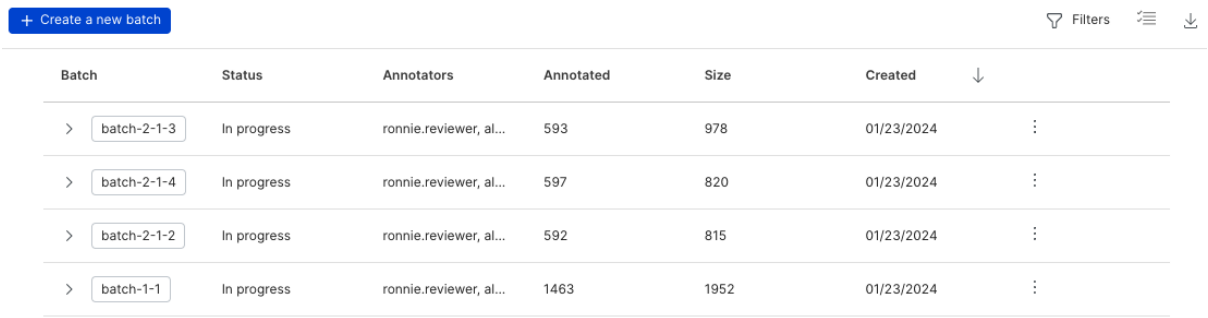
- In the dev split, GT-labeled examples assist with discovering and iterating on labeling functions (LFs).
- In the valid and test splits, use GT-labeled examples to evaluate model performance and facilitate error analysis.

Manage your batches

1. Once create your batches, you can manage the batches in Dataset view from the Batches tab.



2. On the top bar, select the action you want to take:
   - Select the + Create a new batch button to create a new batch of data points for annotation.
   - Select the Filters button to filter the batches by annotator or status.
   - Select the Toggle bulk options icon to show the option to bulk delete batches.
   - Select the Export dataset across all batches icon to export all batches to a CSV file.



3. For individual batches, the following information is available:
   - Date: The date the batch was created.
   - Name: The name of the batch.

- **Label schemas:** The label schema(s) associated with the batch.
- **Batch Size:** The total number of data points in the batch.
- **Annotators:** The list of annotators that have been assigned to annotate the batch.
- **Annotate:** The button to start annotating the batch.

On the top bar, select the action you want to take:

- **Create new batch:** Select the **+ Create a new batch** button to create a new batch of data points for annotation.
- **Search batches:** Use the search bar to search for a specific batch by name.
- **Export data:** Select the **Export dataset across all batches** icon to export all batches to a CSV file.

4. Enter the following options in the **Export annotation batches** modal:
   - **Include columns:** Specify which columns to include in the export. This option enables you to select a focused slice of your data, or you can speed up export time by excluding large columns in a dataset.
   - **Include options:** Specify additional information about individual data points to include such as annotations, comments, slices, filters applied, and model predictions.
   - **Optional settings:** Specify additional options to customize your export. These options include the ability to start an export at a specific index in the dataset, setting a maximum number of data points to export, and options to configure delimiters, quote characters, and escape characters.

Manage individual batches

Select the arrow > icon next to the batch name to view additional information about an individual batch. This menu give you additional actions you can take on an individual batch:

- **Aggregate:** Aggregate annotations from multiple annotators.
- **Commit:** Commit annotations to the ground truth.
- **Set as expert:** Mark selected annotator(s) as an expert for the batch. To mark an individual annotator as an expert for the batch, select the annotator, and then click **Set as expert.** A badge icon shows up next to the annotator's name to indicate the expert status.
- **Remove expert:** To remove the expert designation for an annotator, select **Remove expert**.
- **Edit batch details:** Batch name and annotators list can be updated by clicking on **Edit** icon for a batch.
- **Export batch data:** Export the batch data to a CSV file.
- **Delete batch:** Deletes the batch.

Aggregate annotations

Typically, you'll have more than one annotator reviewing and labeling documents. You can only commit a single vote to ground truth. You can aggregate annotations instead of committing the annotations from a single person. This aggregation helps eliminate potential bias from any particular annotator.

To aggregate annotations, select the annotators that you want to aggregate, then select **Aggregate**.

A majority vote is the only supported aggregation strategy. This strategy takes the majority label for each data point if one exists, and leaves an `UNKNOWN` label where no annotations exist. These results create a new set of annotations with a single vote for each data point. The aggregated annotations can be seen in the expanded view of batch on the **Batches** page, under **Other annotations**.

The majority vote aggregation algorithm works differently for each of these task types.

> NOTE
>
> Snorkel does not support aggregation for free-text label schemas, such as criteria rationale or golden responses. You can commit a single annotator's free-text labels directly.

Single-label

The Snorkel AI Data Development Platform uses a simple majority algorithm as the aggregation strategy for single-label applications. Based on the number of votes for each label, the label with the most votes is applied to the ground truth as the final decision.

For example, consider a single-label application with ten annotators and two classes. Six annotators label a data point with class A, and four annotators label a data point with class B. Because class A now represents the majority vote at aggregation time, the aggregated result will label this data point as class A.

If there is a tie between two or more majority labels, the Snorkel AI Data Development Platform uses a pre-determined random seed to perform a random choice selection.

Multi-label

The Snorkel AI Data Development Platform uses a simple union algorithm as the aggregation strategy for multi-label applications. Based on the number of present/absent votes for each class, the Snorkel AI Data Development Platform applies the label (`present` or `absent`) with the most votes for each class as the final decision.

For example, consider a multi-label application with ten annotators and two classes. For class A, six annotators vote `present` and four annotators vote `absent`. For class B, four annotators vote `present` and six annotators vote `absent`. By taking the majority vote of each class, the aggregated label for the data point applies `present` for class A and `absent` for class B.

If there is a tie between `present` and `absent` labels, the resulting label has an equal chance of being marked as `present` or `unknown`.

Sequence tagging

The Snorkel AI Data Development Platform uses a simple majority algorithm as the aggregation strategy for each character in a span in sequence tagging applications. Based on the number of votes for each label in any interval where annotators disagree, the Snorkel AI Data Development Platform selects the label with the most votes as the final decision.

If there is a tie between two or more majority labels, the resulting label is a negative class to reduce the risk of false positives, such as trailing spaces or mistaken tokens. If the tie is between only positive labels, the resulting label is a pre-determined random seed to perform a random choice selection.

Commit annotations

Once you have a set of annotations that are accurate, commit the annotations as the ground truth for development in Studio.

> NOTE
>
> Only users with the **Developer** or **Administrator** role can commit ground truth.

1. Select the desired annotation set. You can select annotations from either an individual annotator or from an aggregated set of annotations.
2. Select **Commit**.

> ⚠ WARNING
>
> Once you commit annotations, the new annotations overwrite the existing ground truth labels in the data source.

Format for ground truth interaction in the SDK

This section shows example formats for each label space for usage in our SDK for a given datapoint.

Multi-label spaces ground truth is represented as a dictionary where there is a mapping from each label to one of `PRESENT`, `ABSENT`, or `ABSTAIN`:

```
label: {"Japanese Movies": "PRESENT", "World cinema": "ABSTAIN", "Black-and-
White": "ABSENT", "Short Film": "ABSTAIN"}
```

Sequence tagging ground truth for a document is a list of spans, where each span is a triple of (`char_start`, `char_end`, `label`). The spans cannot be empty (`char_start` must be smaller than `char_end`). Overlapping or duplicating spans are not allowed. The sets of char offsets (`char_start`, `char_end`) must be sorted:

```
label: \[ \[0, 29, 'OTHER'\], \[29, 40, 'COMPANY'\], \[40, 228, 'OTHER'\], \
[228, 239, 'COMPANY'\], \[239, 395, 'OTHER'\], \]
```

Single label space ground truth format is represented by their class label

```
label: "loan"
```

How-to: Create dataset views for custom annotation user experiences

Dataset views allow you to customize the presentation of your data for annotators. This makes it easier for annotators to work on a focused set of labeling requirements for a particular annotation batch.

Two types of dataset views are available:

- **Single:** Use this view to display one record at a time to the annotator.
- **Ranking:** Use this view to display multiple records for ranking.

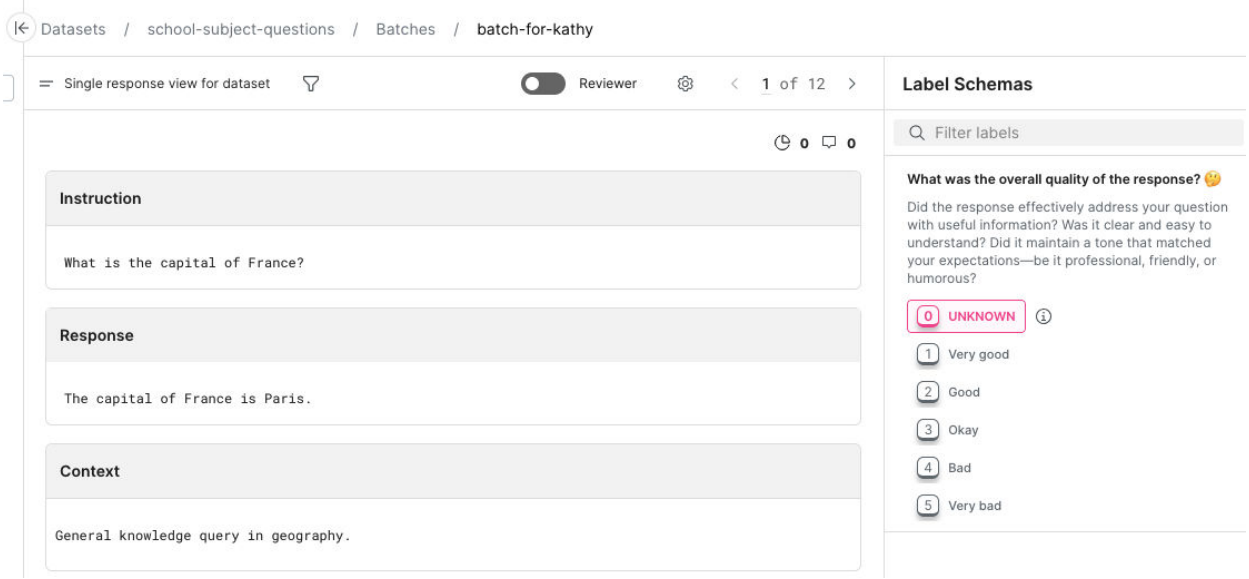
This guide introduces the concepts and the SDK commands for creating dataset views, and also includes two full end-to-end examples and a notebook.

- SDK: Create dataset view
- Example: Create a single response view
- Example: Create a ranking view
- Notebook: dataset-view-creation.ipynb

Single response view

Create a single response data viewer to show the annotator one LLM-generated response at a time. Alongside the response, display one or more label fields for the annotator to fill out. These fields can be multiple choice from a predetermined set of labels, or free text entry.

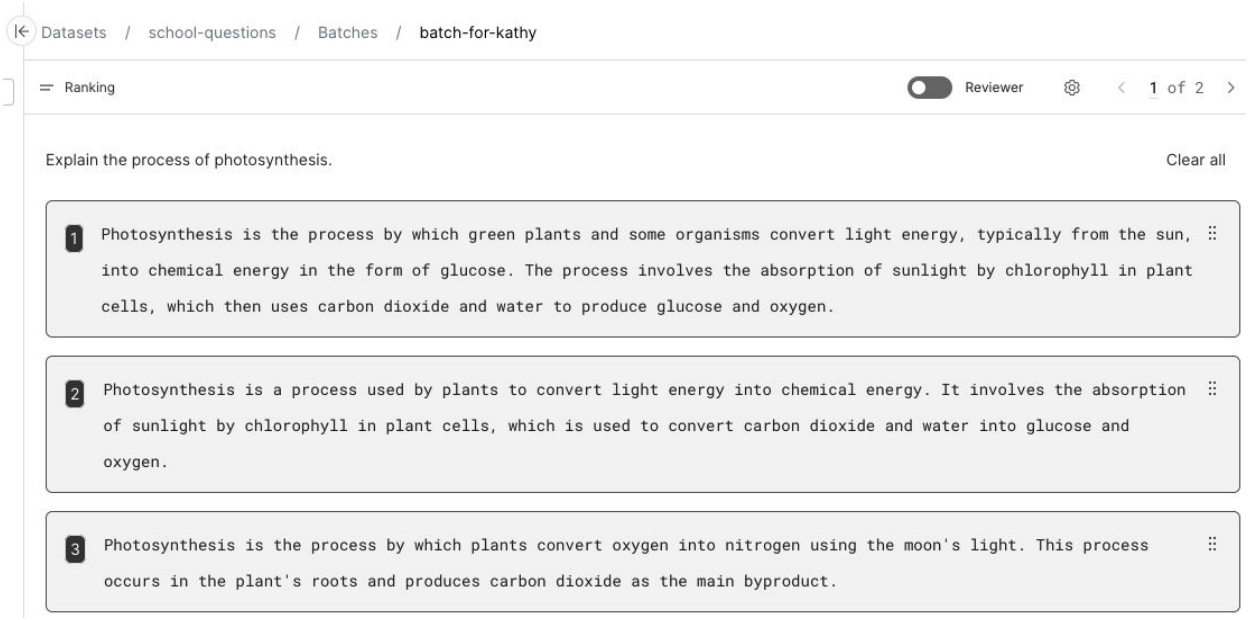
The single dataset view looks like this:



Ranking responses view

Create a ranking data viewer to show the annotator multiple LLM-generated responses to a single prompt. The annotator can drag and drop the responses to rank them from best to worst. Once the annotator orders the responses, each is labeled with the rank number.

The ranking dataset view looks like this:



Prerequisites

- A dataset for which you want to create a view
- A label schema for which you want to create a view. Note that only classification-type label schemas are supported for dataset views.

SDK: Create dataset view

Dataset views require using the Snorkel SDK.

> **NOTE**
>
> The `CONTEXT` column supports a formatted view to display JSON data.

```
# Map the column names in your dataset to the view
column_mapping = {
  FineTuningColumnType.INSTRUCTION: "YOUR_COLUMN_NAME", # Input required, use the column name that has the LLM prompts
  FineTuningColumnType.RESPONSE: "YOUR_COLUMN_NAME", # Input required, use the column name that has the LLM responses
  FineTuningColumnType.CONTEXT: "YOUR_COLUMN_NAME", # Optional, use for notes about the record(s)
  FineTuningColumnType.PROMPT_PREFIX: "YOUR_COLUMN_NAME" # Optional, use the column name that has the system prompt(s)
}
```

```
snorkelflow.create_dataset_view(
  dataset="YOUR_DATASET_NAME", # Input required, enter the existing dataset name
  name="NEW_VIEW_NAME", # Input required, specify a unique view name for this dataset view
  view_type="VIEW_TYPE", # Input required, choose "DatasetViewTypes.SINGLE_LLM_RESPONSE_VIEW" or "DatasetViewTypes.RANKING_LLM_RESPONSES_VIEW"
  column_mapping=column_mapping,
  label_schema_uids=[] # Input required, use the label schema UID you want to use with the view
)
```

Download this code as a notebook.

Example: Create a single response view

Follow this end-to-end tutorial to upload a new dataset, create a label schema, and then create a single response view with an associated batch. This example uses Arcades to illustrate the major steps. Arcades are lightly interactive annotated screenshots that you can click through to see the steps in action.

Upload a dataset

1. Download the school-subject-questions.csv dataset.
2. In the Snorkel AI Data Development Platform, select **Upload dataset**.
   - **Dataset name:** `school-subject-questions`.
   - **Enable multi-schema annotations:** Select this checkbox.
   - **File Upload:** Select this option.
   - **Choose File:** Choose the `school-subject-questions.csv` file.
   - **Split:** Select the `train` split.
3. Select **Verify data source(s)**.
   - **UID Column:** Select `UID` from the dropdown.
   - **Data type:** Select **Raw text**.
   - **Text type:** Select **Classification**.
   - **Primary text field:** Select `Response`.
4. Select **Add data source(s)**.

Wait for the dataset to upload. You should see **Data ingestion complete** and the name of the dataset.

Click through the Arcade below to see these steps in action.

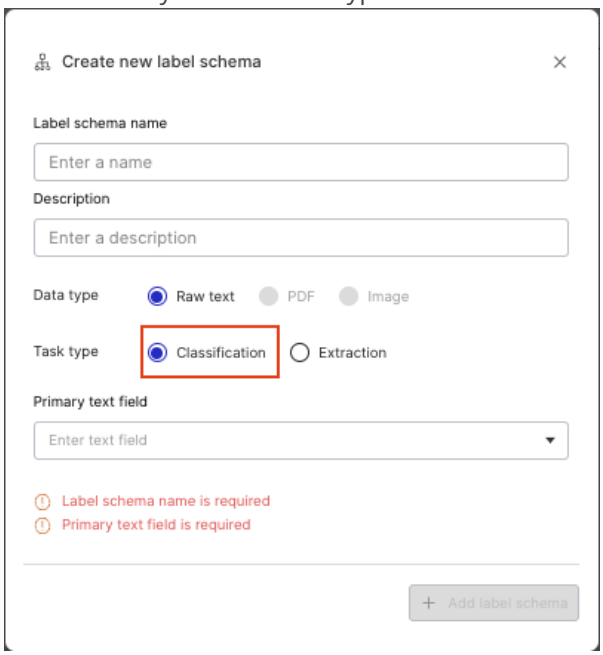
Visit the Snorkel docs site to interact with this element

Create a label schema

1. From the dataset page, select **Label Schemas**.

2. Select **Create new label schema**.
   - **Label schema name:** Enter a name for the label schema.
   - **Description:** Enter a description for the label schema.
   - **Data type:** Select **Raw text**.
   - **Task type:** Select **Classification**.
   - **Primary text field:** Select `Response`.
3. Select **+ Add new label**. Enter as many labels as you want to use for the label schema.
4. Select **Add label schema**.

> NOTE
>
> Note that only classification-type label schemas are supported for dataset views.



Click through the Arcade below to see these steps in action.

Visit the Snorkel docs site to interact with this element

Create a single view using the SDK

1. Download the dataset-view-creation.ipynb notebook.
2. In the Snorkel AI Data Development Platform, select **Notebook**.
3. Select **Upload**.
4. Upload the notebook.
5. Select `dataset-view-creation.ipynb` to open it.
6. Run the notebook to create a single response view. Many cells require user input. Make note of the following:

- `dataset_name = "school-subject-questions"`

- `view_name = "Single Response View"`: You can create any name you want.
- `view_type= "DatasetViewTypes.SINGLE_LLM_RESPONSE_VIEW"`
- `FineTuningColumnType.INSTRUCTION: "Instruction"`
- `FineTuningColumnType.RESPONSE: "Response"`
- `FineTuningColumnType.CONTEXT: "Context"`
- `FineTuningColumnType.PROMPT_PREFIX: ""`: This line is optional.
- `label_schema_uids=[XXXX]`: Your UID will vary.

7. When you have finished running the notebook, return to the Snorkel AI Data Development Platform.

Next, create a batch that uses the view.

Click through the Arcade below to see these steps in action. Note that the notebook may not match the Arcade exactly, and you must create the batch before you can annotate.

Visit the Snorkel docs site to interact with this element

Create a batch for annotation

1. From the dataset page, select **Batches**.
2. Select **Create new batch**.
   - **Batch name:** Enter a name for the batch.
   - **Label schemas:** Select the label schema you created.
   - **Split:** Select `train`.
   - **Assign to:** Select an annotator.
3. Select **Create batch**.

Click through the Arcade below to see these steps in action.

Visit the Snorkel docs site to interact with this element

Annotate using the single dataset view

1. From the batch page, use the two-line menu in the top left of the batch, and choose the view that matches the `view_name` that you created earlier in the notebook.

2. Now you can explore the dataset view from the perspective of the annotator. Make adjustments to the view as needed.

Click through the Arcade below to see these steps in action.

Visit the Snorkel docs site to interact with this element

Example: Create a ranking view

Follow this end-to-end tutorial to upload a new dataset, create a label schema, and then create a ranking view with an associated batch. This example uses Arcades to illustrate the major steps. Arcades are lightly interactive annotated screenshots that you can click through to see the steps in action.

Upload a dataset

1. Download the school-subject-questions.csv dataset.
2. In the Snorkel AI Data Development Platform, select **Upload dataset**.
   - **Dataset name:** `school-subject-questions`.
   - **Enable multi-schema annotations:** Select this checkbox.
   - **File Upload:** Select this option.
   - **Choose File:** Choose the `school-subject-questions.csv` file.
   - **Split:** Select the `train` split.
3. Select **Verify data source(s)**.
   - **UID Column:** Select `UID` from the dropdown.
   - **Data type:** Select **Raw text**.
   - **Text type:** Select **Classification**.
   - **Primary text field:** Select `Response`.
4. Select **Add data source(s)**.

Wait for the dataset to upload. You should see **Data ingestion complete** and the name of the dataset.

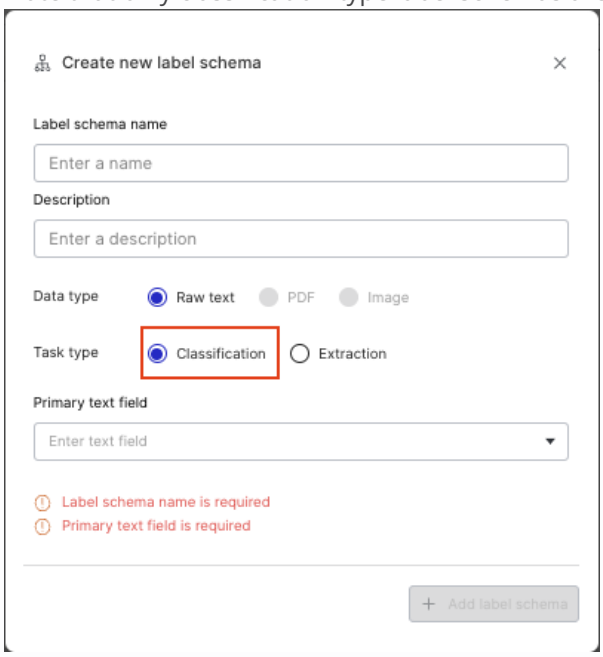
Click through the Arcade below to see these steps in action.

Visit the Snorkel docs site to interact with this element

Create a label schema

1. From the dataset page, select **Label Schemas**.
2. Select **Create new label schema**.
   - **Label schema name:** Enter a name for the label schema.
   - **Description:** Enter a description for the label schema.
   - **Data type:** Select **Raw text**.
   - **Task type:** Select **Classification**.
   - **Primary text field:** Select `Response`.
3. Select **+ Add new label**. Enter numeric labels from 1 to 3.
4. Select **Add label schema**.

> NOTE
>
> Note that only classification-type label schemas are supported for dataset views.



Click through the Arcade below to see these steps in action.

Visit the Snorkel docs site to interact with this element

Create a ranking view using the SDK

1. Download the dataset-view-creation.ipynb notebook.
2. In the Snorkel AI Data Development Platform, select **Notebook**.

3. Select **Upload**.
4. Upload the notebook.
5. Select `dataset-view-creation.ipynb` to open it.
6. Run the notebook to create a ranking response view. Many cells require user input. Make note of the following:

- `dataset_name = "school-subject-questions"`
- `view_name = "Ranking View"`: You can create any name you want.
- `view_type= "DatasetViewTypes.RANKING_LLM_RESPONSES_VIEW"`
- `FineTuningColumnType.INSTRUCTION: "Instruction"`
- `FineTuningColumnType.RESPONSE: "Response"`
- `FineTuningColumnType.CONTEXT: "Context"`
- `FineTuningColumnType.PROMPT_PREFIX: ""`: This line is optional.
- `label_schema_uids=[XXXX]`: Your UID will vary.

7. When you have finished running the notebook, return to the Snorkel AI Data Development Platform.

Next, create a batch that uses the view.

Click through the Arcade below to see these steps in action. Note that the notebook may not match the Arcade exactly, and you must create the batch before you can annotate.

Create a batch for annotation

1. From the dataset page, select **Batches**.
2. Select **Create new batch**.
   - **Batch name:** Enter a name for the batch.
   - **Label schemas:** Select the label schema you created.
   - **Split:** Select `train`.
   - **Assign to:** Select an annotator.
3. Select **Create batch**.

Click through the Arcade below to see these steps in action.

Annotate using the ranking dataset view

1. From the batch page, use the two-line menu in the top left of the batch, and choose the view that matches the `view_name` that you created earlier in the notebook.

2. Now you can explore the dataset view from the perspective of the annotator. Make adjustments to the view as needed.

Click through the Arcade below to see these steps in action.

Visit the Snorkel docs site to interact with this element

Notebook: dataset-view-creation.ipynb

View and download the dataset-view-creation.ipynb notebook:

Visit the Snorkel docs site to interact with this element

Using multi-schema annotations

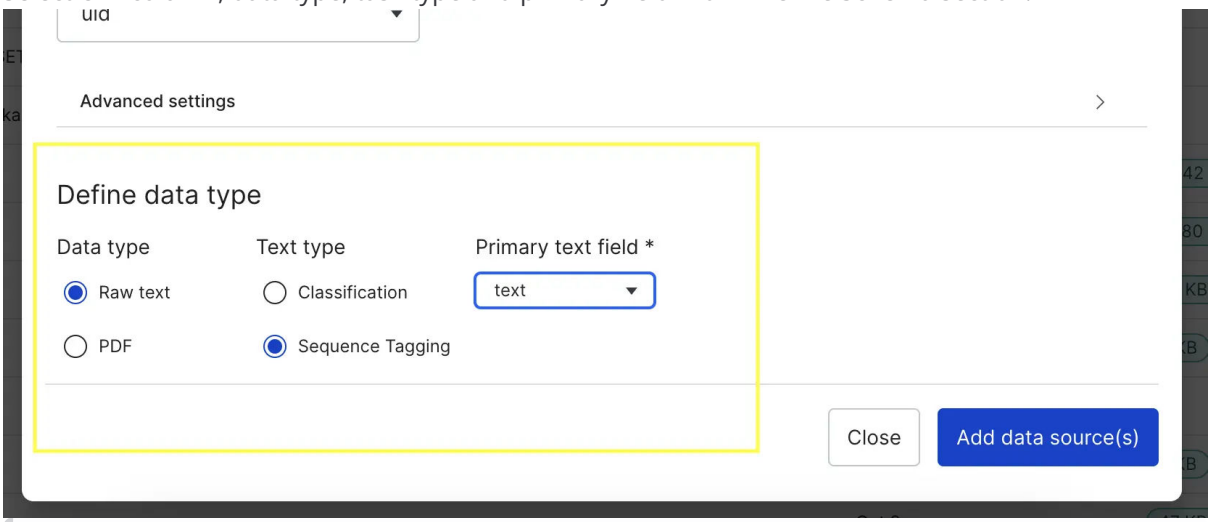
This article explains how to use multi-schema annotations, including uploading a multi-schema annotation dataset, annotating multiple schemas, and reviewing the annotations and progress.

Multi-schema annotations empowers subject matter experts to work more efficiently. This feature lets you collect annotations across multiple schemas at one time, unlocking complex workflows. With multi-schema annotations, datasets become the new home for all of your annotations and ground truth (GT). The GT is stored in a label schema for a dataset, which can be used by all of the downstream model nodes.

By default, all text-only datasets are multi-schema annotations. Multi-schema annotations are not supported for PDF or image datasets.

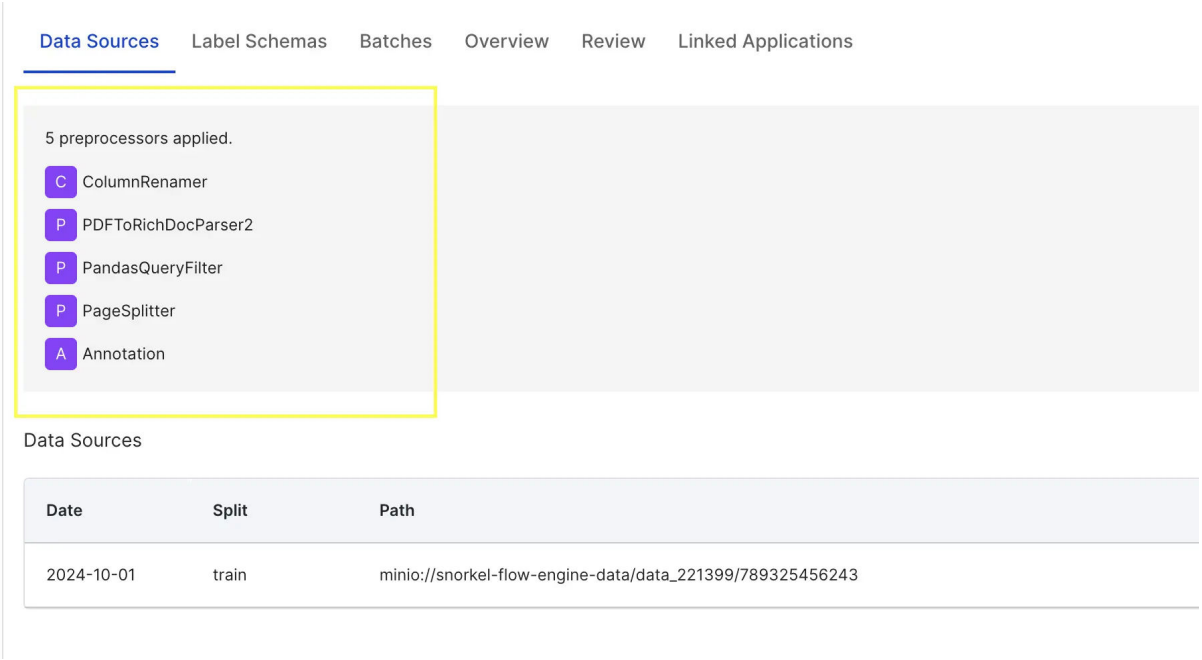
Upload a multi-schema annotation dataset

1. To create a new dataset, select **Datasets > Upload new dataset**.
2. Enter the required information for creating your dataset. For more, see Uploading a dataset.
3. Select **Verify data source(s)**.
4. Select UID column, data type, task type and primary field within Define Schema section.



BASED ON THE DATA TYPE YOU SELECT, THE OPTIONS FOR TASK TYPE AND primary field may change. For supported data types, we'll automatically pre-process your data to make it easier for you to work with during annotation. :::

5. Once the data sources are uploaded, you'll see the applied pre-processors in the **Data sources** tab.

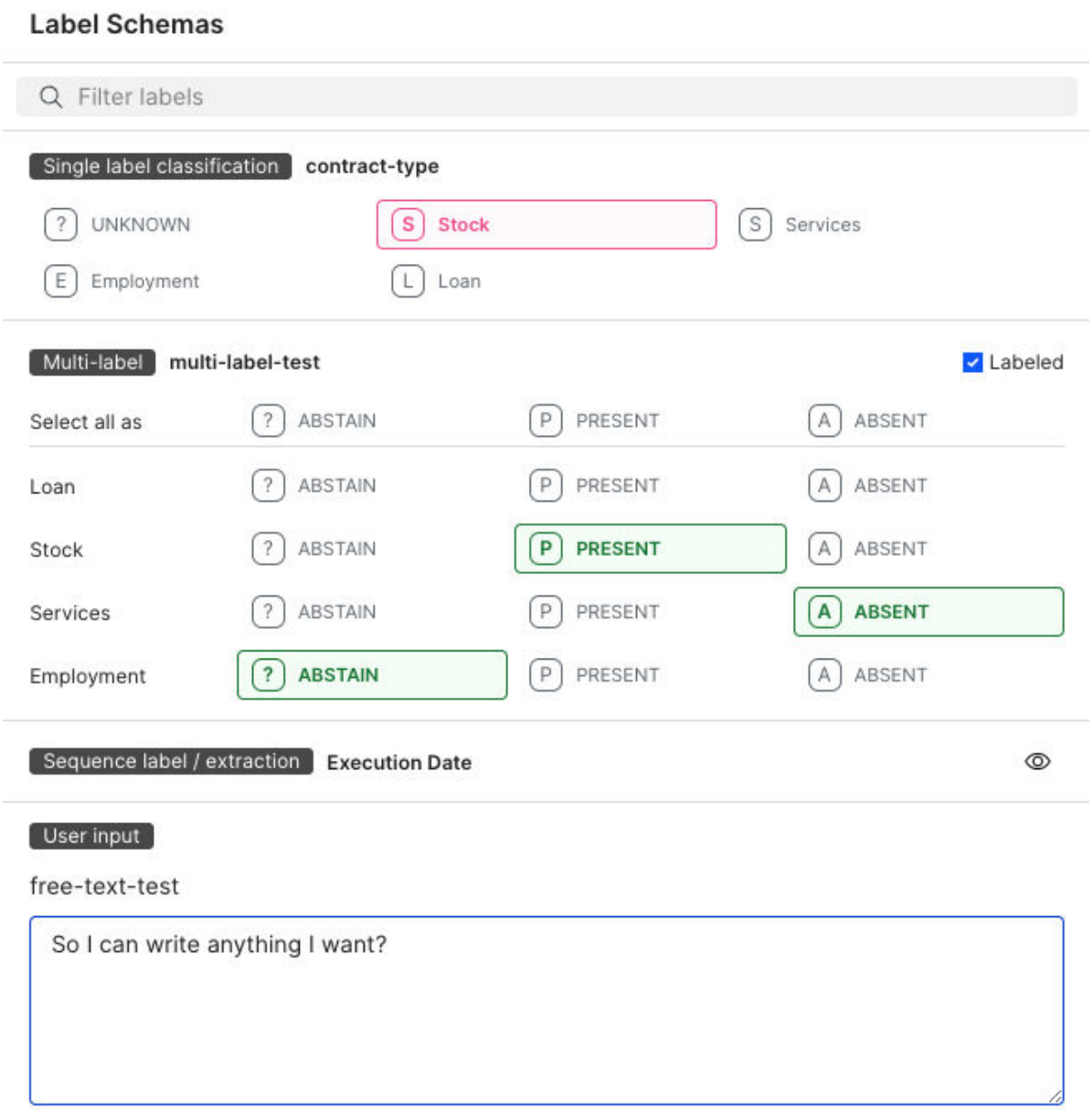


6. Within **Datasets > "your dataset name" > Label Schemas**, select **+ Create new label schema**.
7. Enter a name, description, data type, task type, and additional fields for each task type.
   - Classification tasks:
     1. Select **Classification** as the **Task type**.
     2. Select **Text** as the **Primary text field**.
     3. Select **Single label**, **Multi-label**, or **Text label**. **Text label** allows for free text in your labels instead of a defined label or labels for the other options.
   - Extraction tasks:
     1. Select **Extraction** as the **Task type**.
     2. Select **Sequence tagging**.
     3. Select the main text field as the **Primary text field**.
     4. Define your **IOU Agreement Threshold**, which is the percentage of words that overlap to count as an agreement in the IAA matrix.
     5. For **Label type**, select **Single label** for spans that cannot overlap or **Overlapping (annotation-only)** for spans that can overlap.
     6. Select **Add new label** to define new label options along with a description.
8. Select **+ Add label schema**.
9. In **Batches**, select **+ Create new batch.**
10. Enter the batch name.
11. Select your split and your label schema.
12. Enter your batch numbers and batch sizes.
13. (Optional) Assign users to annotate the batch.
14. Select **Create batch**.

Annotate multiple schemas

The Snorkel AI Data Development Platform applies annotations to the data points across any batch in which the annotations are used.

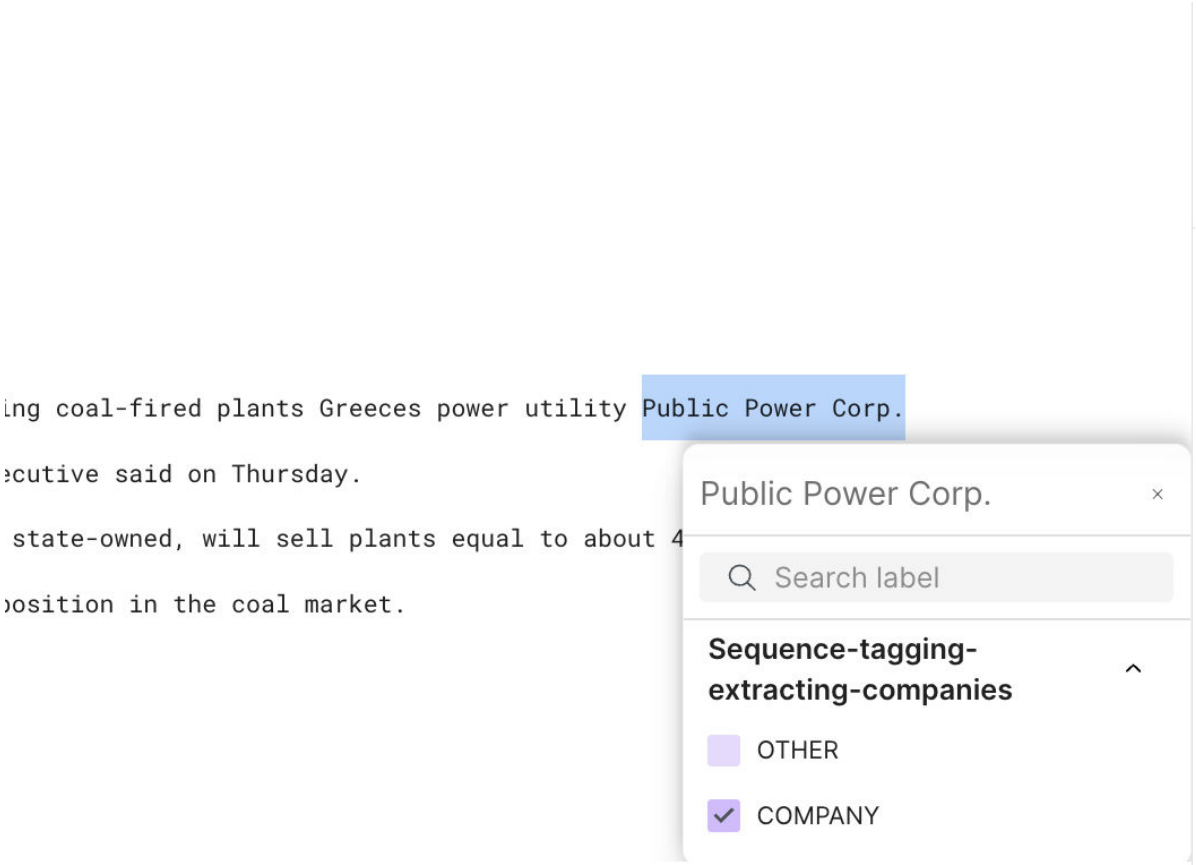
1. In the **Batches** tab, select **Annotate** beside the batch you want to annotate.
2. Select the labels that apply to the data point.



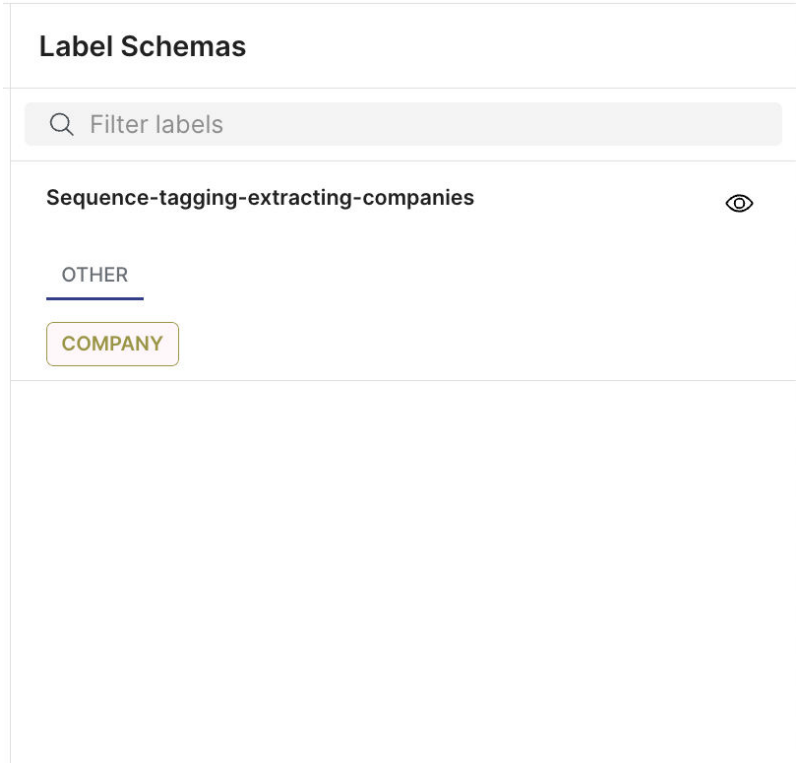
3. Select the previous and next arrows to move between data points.
4. Continue annotating each data point until you have completed your annotations.

Annotate sequence tagging

The Snorkel AI Data Development Platform supports sequence tagging for extraction tasks. Spans are key pieces of information that you want to extract from a document. To label spans in the document, you can highlight a section of text and select the span label from the pop-up menu.



If you want to apply the same label to a series of spans, first select the label from the right-side menu, and then highlight all of relevant the text segments in the document.

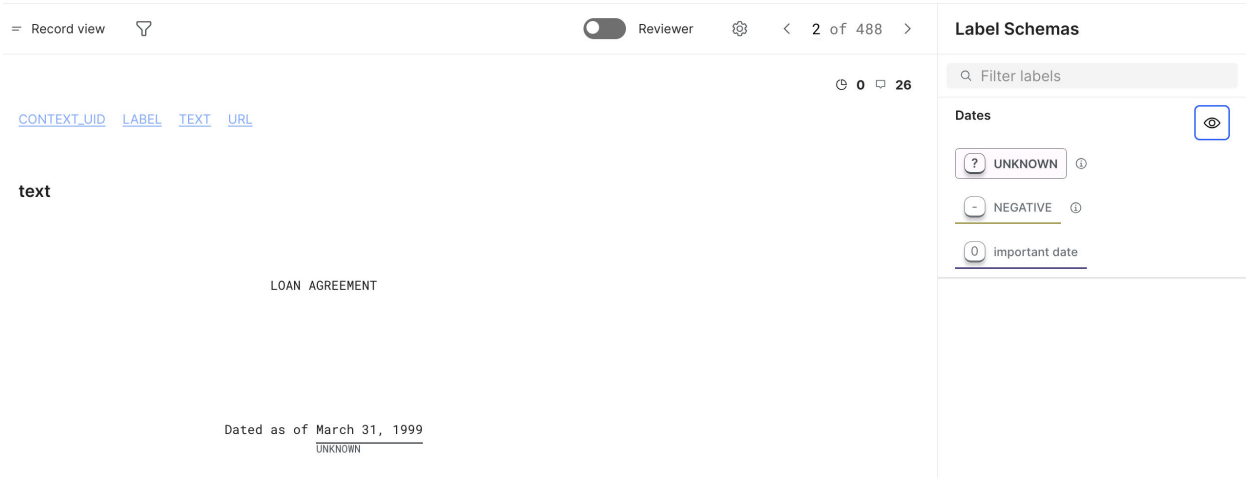


If you selected **Single label** during the label schema creation process, the spans cannot overlap. If you select **Overlapping (annotation-only)**, you can use overlapping spans.

Annotate candidate extraction

Similar to sequence tagging, candidate extraction (CE) involves annotating spans in the text. The key difference is that CE schemas are pre-defined based on an extractor, such as a regex. To label these spans, click on a highlighted schema and use the sidebar to choose the label.

You can also use keyboard shortcuts to navigate through the spans. Right and left arrows change the documents while the top and bottom arrows move through the spans within a document. Keyboard shortcuts corresponding to the labels in the sidebar, which can be used to label a selected span.

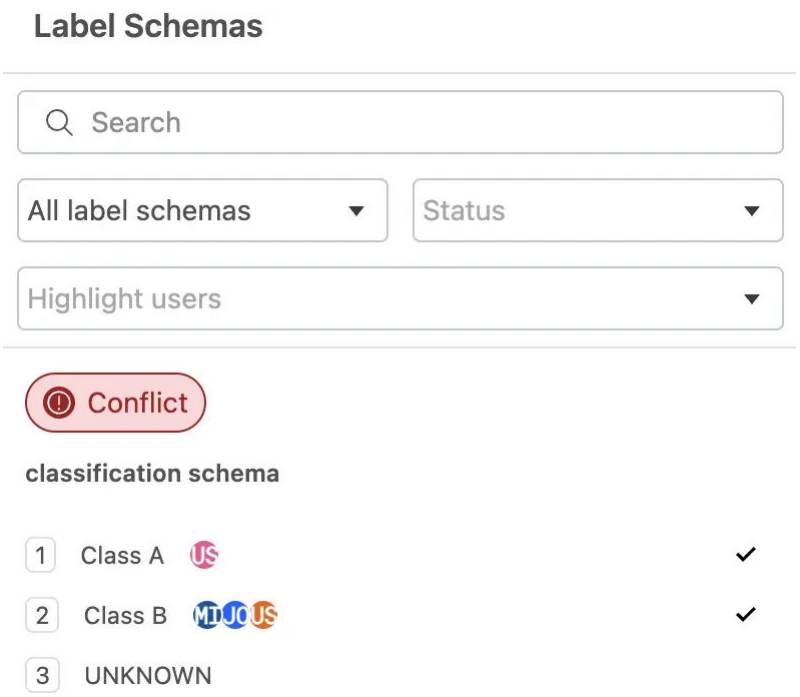


Review annotations

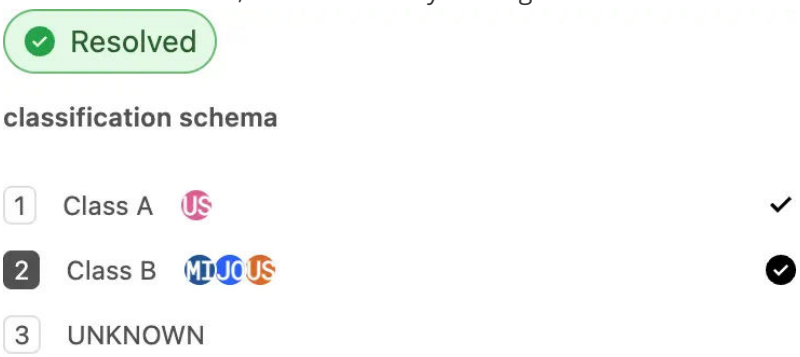
If you have **Reviewer** permissions, enable **Reviewer** mode during annotation to reconcile your team's annotations and select the specific ones to commit to ground truth.

1. Enable **Reviewer** mode with the toggle to see all of the annotations for a dataset. You will see the annotations from each annotator.

2. Hover over the circles to see which user annotated each specific class. Alternatively, use the filters for the **Label Schemas** to filter down to a specific label schema, conflict status, or user.



3. If there are conflicts, resolve them by clicking the checkmark next to the correct class.



4. Once all the conflicts have been resolved, select the **Commit** button to commit the selected annotations to ground truth for this data point.

Commit annotations in bulk

Alternatively, you can commit the annotations in bulk:

1. Select **Datasets** in the left navigation.
2. Select a dataset.
3. Select the **Batches** tab.
4. Expand a batch.
5. Select an annotator or aggregated source.
6. Select **Commit**.

> **NOTE**
>
> - Note that when you make a ground truth commit, it overwrites any existing ground truth in your dataset.
> - Reviewer mode doesn't support committing ground truth for multi-label classification and text label schemas. Use this batch workflow instead.

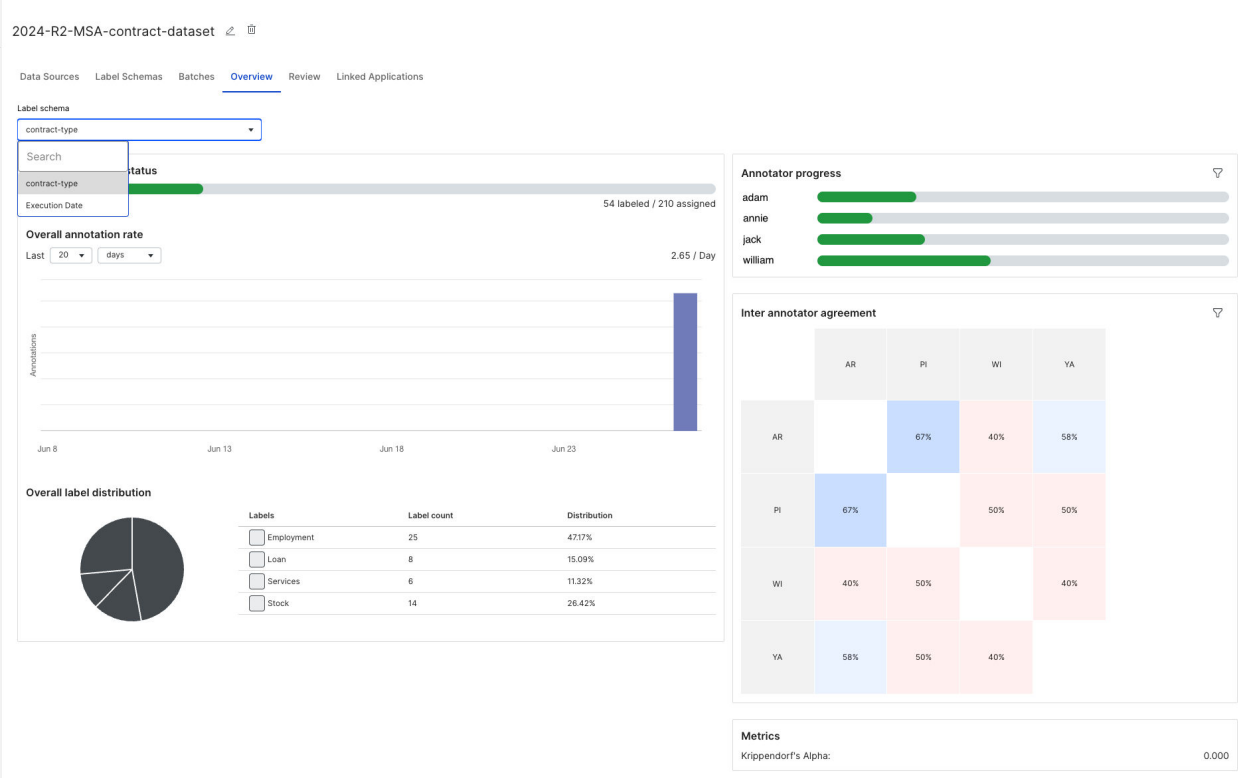
Configure your annotation display

1. In the **Batches** tab, select **Annotate** beside the batch you want to annotate.
2. If you want to filter the data points to annotate, select your filtering options with fields and operators.
3. To change your display settings, select the gear icon. You can change the displayed columns, column order, and text direction. You can also prioritize unlabeled documents and set a default multi-label class.

View annotation progress

In the **Overview**, you can select your **Label schema** from the dropdown menu to see the current status and how much each annotator has finished.

You can select filters for **Annotator progress** to see the progress for specific annotators.



You can also see the **Inter annotator agreement**, which represents how often annotators agree with each other. For sequence tagging label schemas, agreement is defined during the label schema creation process, where the **IOU Agreement Threshold** defines what percentage of words have to overlap to count as an agreement in the IAA matrix. This defaults to 100%.

In the **Review** tab, you can see that annotators, annotations, and batches.

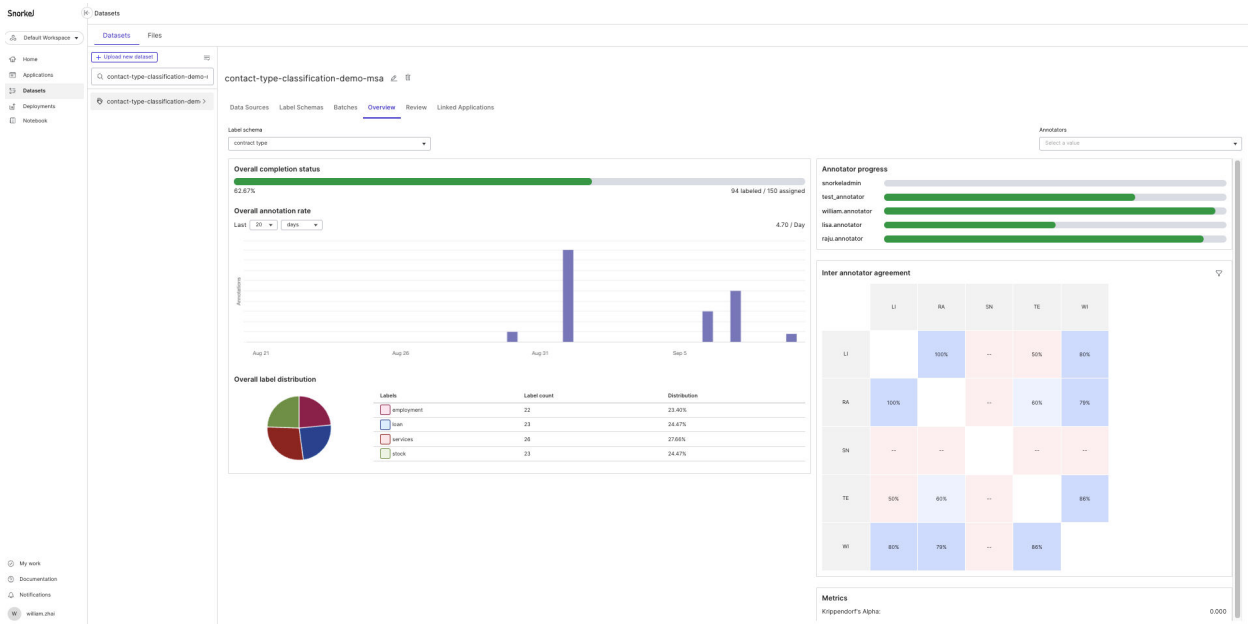
Overview page: View aggregate annotation metrics

This page walks through the summary metrics and settings that are found on the Datasets **Overview** page. This page is typically used by those with Developer and Reviewer roles that want a broad view of the annotations that have been completed as well as insight into the level of agreement among annotators.

Overview page

The **Overview** page shows you various aggregate metrics on the number of annotations that have been completed, the distribution of labels, annotator agreement, and a view into recent annotator activity.

You can select the **Filter** button on the top-left corner of the page to filter data by label schema, allowing you to view metrics specific to a particular schema. Additionally, you can use the **Filter** button on the top-right corner of the screen to filter charts by specific annotators.



Overall completion status

The **Overall completion status** chart displays the percentage of assigned datapoints that have been annotated. The progress bar gives a good visual representation of how much annotation progress has been made.

Overall annotation rate

The **Overall annotation rate** bar chart displays the aggregate number of annotations that have been completed at different granularities that you can specify (e.g., last 20 days, last 4 months).

Calendar boundaries are used when referencing days (starting at 12:00am), weeks (starting Monday), and months (starting on the 1st day of the month). For example, viewing annotations from the last 2 weeks on a Wednesday will show all annotations that have been completed since two Mondays ago, not 14 days ago. Similarly, the last X months view will show all annotations that occurred within each corresponding month.

All annotations are displayed relative to your timezone, which is displayed in the chart. To change your timezone, select your user name in the bottom left corner of your screen. Click **User settings**, then click **Advanced**. Note that the SDK retrieves annotation timestamps in the UTC timezone, which is how timestamps are stored in the database.

Overall label distribution

The **Overall label distribution** chart displays the distribution of labels that have been annotated. This chart enables you to quickly see if there are particular classes that are underrepresented in the annotations that have been completed so far. Users with developer access can take this information and create new batches to get a more even distribution of labels. See Create batches for more information on how to create new batches.

Annotator progress

The **Annotator progress** chart displays the percentage of assigned datapoints that have been annotated, split up by each annotator. Hover over the progress bars for each annotator to see exactly how many datapoints that they have annotated compared to the number of datapoints that have been assigned to them.

Inter-annotator agreement

The **Inter-annotator agreement** chart displays the agreement rate between annotators where they have labeled the same data points. For example, if annotators A and B have annotated 10 of the same data points, and on 8 of those they have the same labels, then the shared grid will show 80% in the corresponding cell.

You can filter the chart by label class and or by batch. To do so, click the **Filters** button at the top right corner of the chart, then select your desired filters.

The **Inter-annotator agreement** chart will only show up to 10 annotators at a time to ensure a consistent, readable experience. To view the chart for all annotators, click the chart to render a modal of the full chart that is not truncated.

Below the chart you can find the value for Krippendorff's Alpha. This metric measures disagreement among annotators, corrected for disagreement that is expected from random chance. Ideally you want to see a value closer to 1, which indicates better agreement among annotators.

(Beta) Using word-based annotations

> NOTE
> This is a beta feature available to customers using a Snorkel-hosted instance of the Snorkel AI Data Development Platform. Beta features may have known gaps or bugs, but are functional workflows and eligible for Snorkel Support. To access beta features, contact Snorkel Support to enable the feature flag for your Snorkel-hosted instance.

The Snorkel AI Data Development Platform supports word-based annotations within PDFs, specifically for named entity recognition (NER). You can annotate a single word or a group of words based on your annotation task, such as a person, location, or organization, etc.

Follow the steps outlined below to add and remove word-based annotations in Annotation Studio.

Annotating a single word

Follow these steps to annotate a single word in your PDF.

Visit the Snorkel docs site to interact with this element

Annotate a single word

1. Navigate to the word you want to annotate.
2. Double-click on the word to select the word and open the annotation menu.
3. Choose the correct label from the annotation menu.

The word is now annotated with the label you selected.

Remove annotation from a single word

1. Navigate to the annotation you want to remove.
2. Single- or double-click on the annotated word to select the word and open the annotation menu.
3. Hover over the selected label and click on the "x" icon to remove it.

The annotation is now removed from the word.

Annotating multiple words

Follow these steps to annotate multiple words in your PDF.

Visit the Snorkel docs site to interact with this element

Annotate multiple words

1. Navigate to the section containing the group of words you want to annotate.

2. Click and hold your mouse at the starting point of the desired text, then drag to create a bounding box around the group of words.
3. Release the mouse button to highlight the selected text and open the annotation menu.
4. Choose the correct label for the selected text from the annotation menu.

The group of words is now annotated with the label you selected.

Remove annotation from multiple words

1. Navigate to the section containing the group of words from which you want to remove the annotation.
2. Click and hold your mouse at the starting point of the desired text, then drag to create a bounding box around the group of words.
3. Release the mouse button to highlight the selected text and open the annotation menu.
4. Hover over the selected label and click on the "x" icon to remove it.

The annotation is now removed from the group of words.

Prompt development overview

Prompt development, sometimes called prompt engineering, is the process of designing and refining inputs to guide AI models, such as large language models (LLMs), to produce high-quality, task-specific outputs. In the Snorkel AI Data Development Platform, prompt development workflows empower users to create, experiment with, and optimize prompts iteratively. This ensures efficient, precise, and reproducible outcomes for downstream tasks.

A prompt development workflow begins with uploading a text dataset, selecting an LLM, and crafting system and/or user prompts to tailor the model's response. With prompt versioning, users can iterate on their designs, enabling continuous improvement of AI-driven results.

Prompt development enables users to harness the full potential of AI models by guiding them to produce accurate, efficient, and contextually relevant outputs. By optimizing prompts, it reduces the need for extensive fine-tuning, accelerates workflows, and ensures models align with specific business goals or user needs.

Components of prompt development

Input dataset

A prompt development workflow uses an input dataset. This dataset serves as context or examples for the LLM to work with. An input dataset for prompt development should contain relevant data points or scenarios that the LLM can use to align its output with your goals. One example of an input dataset is question and context pairs that can be used for chatbot development by providing an LLM with a prompt and the context needed to answer domain-specific questions.

Best practices

When creating input datasets, it is important to follow best practices:

- Use clear column names that map to LLM processes and predictions. For example, `User Input`, `Context`, and `Expected Response`.
- Make sure you **enable multi-schema annotations** when uploading the input dataset.
- Make sure you select the **train split** when uploading the input dataset.

> NOTE
>
> Before creating a prompt development workflow, upload the dataset you plan to use. Learn more about uploading datasets.

LLMs

The prompt development workflow allows you to experiment with different LLMs to identify the model and prompt combination that delivers the highest quality results.

Prompts

There are multiple types of prompts in prompt development:

- **System prompt:** Sets the overarching tone, rules, or constraints for the LLM. Examples include:
  - Instructing the model to adopt a persona. `You are a financial assistant.`

  - Ensuring responses adhere to specific styles or formats. `Format this using the Google Developer Style Guide.`

- **User prompt:** Focuses on the primary task or request. These prompts add task-specific instructions, often referencing input dataset columns dynamically. Examples include:
  - For **simple tasks** not requiring global rules or a consistent persona. `Translate the following text into French: {Text}.`
  - When the **task or output is fully defined** in the user prompt. `Extract all ingredients from the following recipe: {Recipe}.`

Using multiple prompts together

While user prompts can suffice for simple tasks, combining system and user prompts is recommended for:

- **Consistency:** The system prompt ensures predictable behavior. For example:
  - System prompt: `You are a data assistant summarizing technical documents.`
  - User prompt: `Summarize the following text: {Text}.`
- **Complex workflows:** System prompts provide global instructions while user prompts handle granular tasks. For example:
  - System prompt: `You are an expert in natural language processing. Always respond in JSON format.`
  - User prompt: `Extract named entities from the following text: {Text}.`

Features and benefits

Key features

- **Workflow management:** Create, view, and manage prompt workflows via the Prompts page.
- **Dataset selection:** Choose datasets with clearly displayed columns.
- **Prompt experimentation:** Test different LLMs, system prompts, and user prompts.
- **Prompt versioning:** Save, view, and compare prompt iterations.
- **Prompt version favoriting:** Star your favorite prompt versions.
- **Prompt version renaming:** Rename prompt versions to help track different iterations.
- **SME feedback:** You can create batches for SME annotation, and view ground truth annotations provided by SMEs on input data and LLM responses for each data point, directly from the **Develop prompt** page.
- **Multiple views:** View your input data, LLM responses, and annotations in a table or one data point at a time.
- **Filter views:** Filter your input data, LLM responses, and annotations by ground truth, by slice, and by input field value.
- **Selective runs:** Run prompts on a subset of your data to enable faster iteration.
- **Export:** Export the LLMAJ output dataset and/or prompt template (including model, system prompt, and user prompt) for any prompt version and run.
- **Enhance prompt with ground truth:** You can add ground truth annotations to your prompt to help guide the LLM to produce more accurate responses.

Workflow capabilities

1. **Run prompts:** Execute prompts and view LLM responses.
2. **Iterative refinement:** Update prompts iteratively for continuous improvement.
3. **Comparison view:** Analyze multiple LLM responses for different prompt versions, side-by-side.
4. **Incorporate SME feedback:** Request and view subject matter expert annotations to improve prompts.

Known limitations

These are the known limitations for prompt development:

- Only **text datasets** are supported.
- Multi-schema annotations (MSA) must be enabled when uploading the input dataset.
- Datasets can be up to 420MB in size.
- Train, test, and valid splits are supported, but are combined into a single dataset for all runs.
- Only local download is supported for exporting the prompt run output dataset and prompt template files.

What's next?

You can learn to create a prompt development workflow.

Create prompt development workflow

Prompt development, sometimes called prompt engineering, is the process of designing and refining inputs to guide AI models, such as large language models (LLMs), to produce high-quality, task-specific outputs. A prompt development workflow begins with uploading a text dataset, selecting an LLM, and crafting system and/or user prompts to tailor the model's response. With prompt versioning, users can iterate on their designs, enabling continuous improvement of AI-driven results.

Prerequisite

Enable required models via the **Foundation Model Suite** to set up and manage external models. Learn more about using external models.

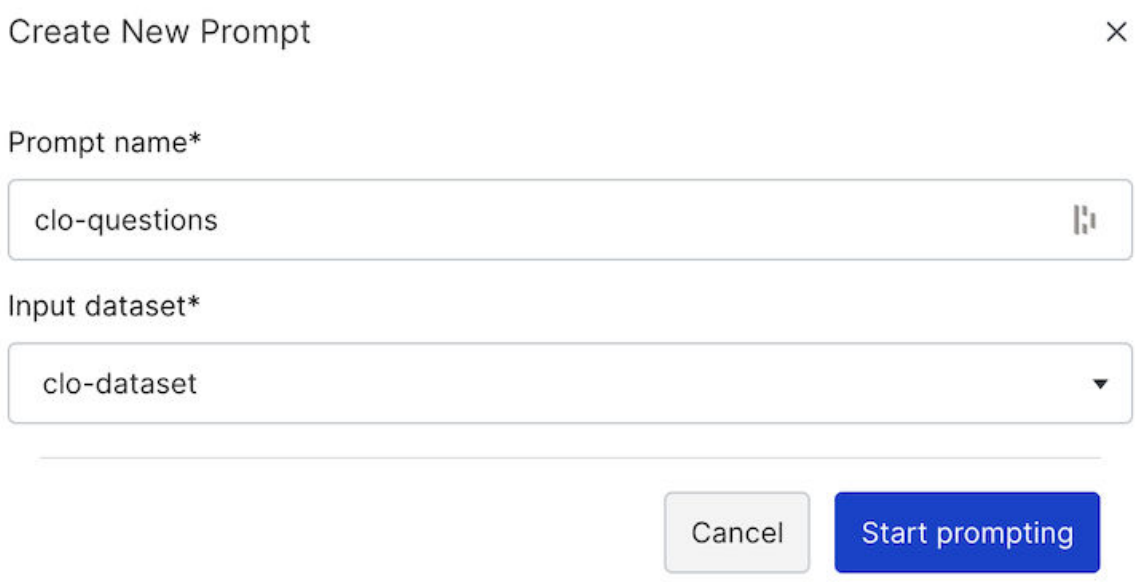
Create a prompt development workflow

Upload input dataset

1. Navigate to the **Datasets** page.
2. Select **Upload new dataset**.
3. Select the train split when uploading.

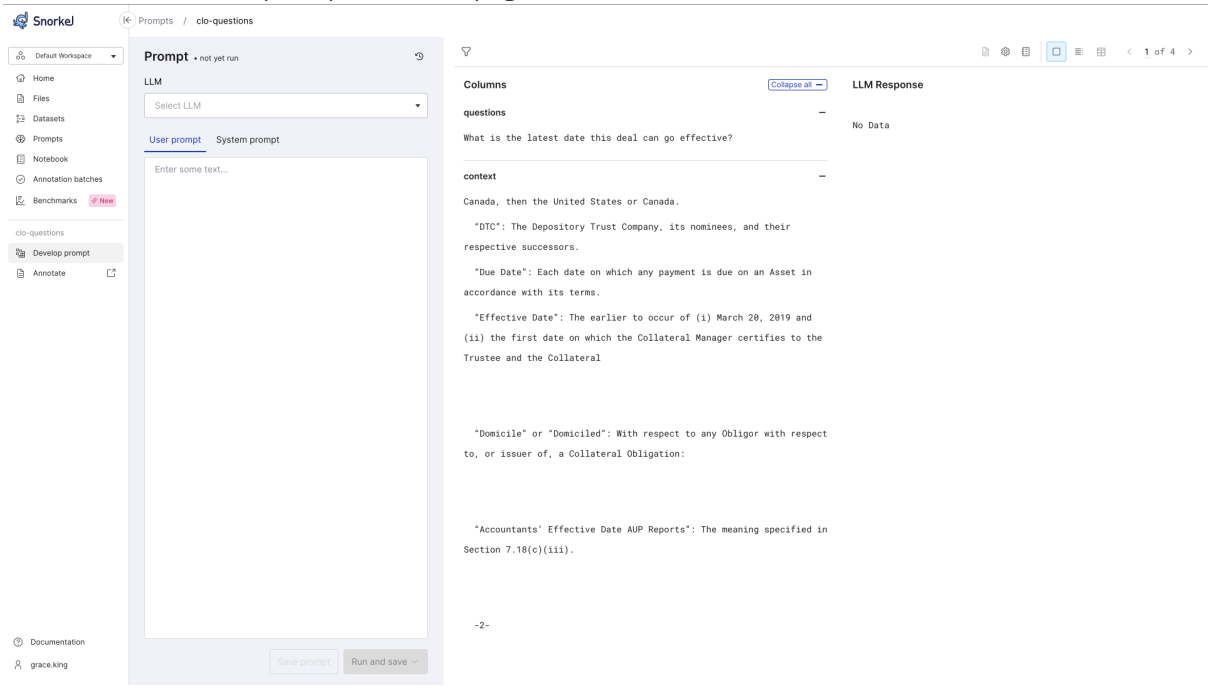
Create a prompt workflow

1. Go to the **Prompts** page.
2. Select **Create Prompt**.



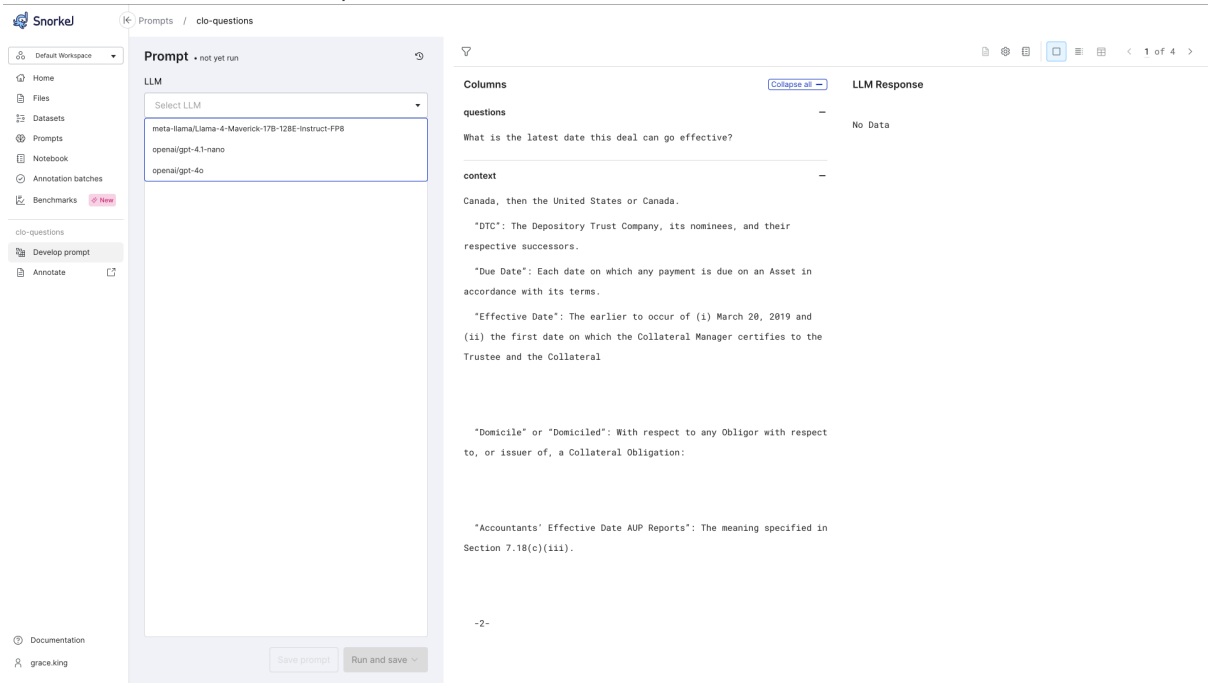
3. Name your workflow.
4. Associate it with an input dataset.

You will see the initial prompt workflow page:



Select model

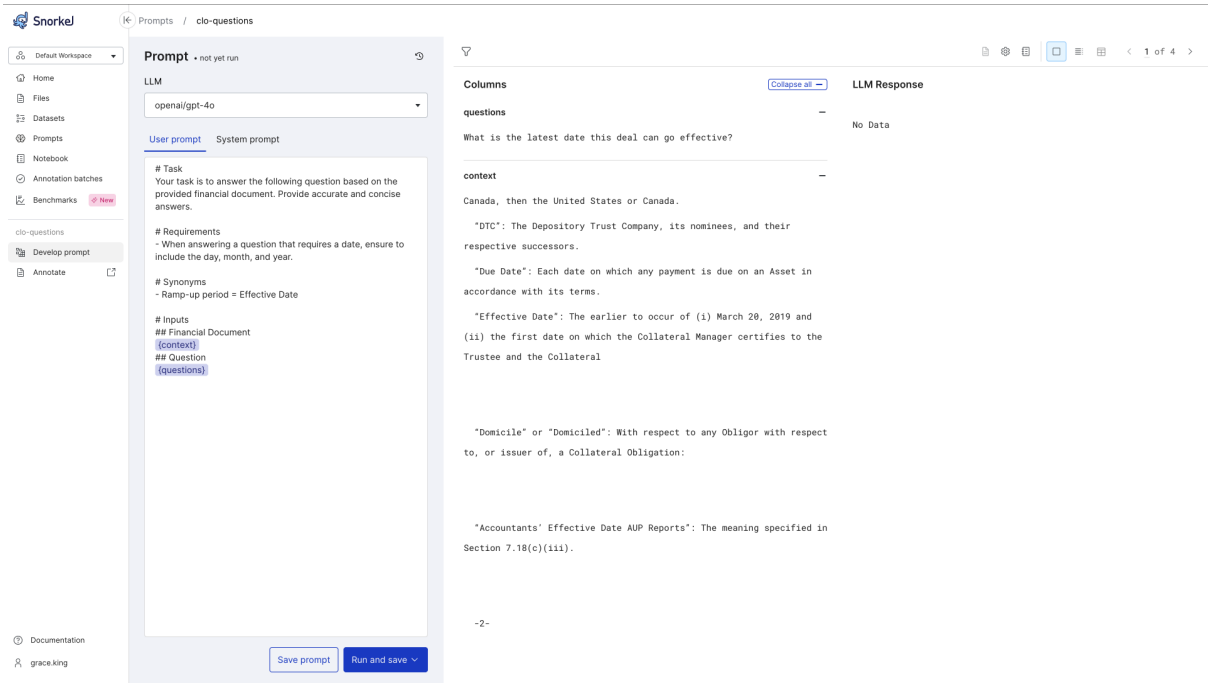
1. Choose an LLM from the dropdown menu.



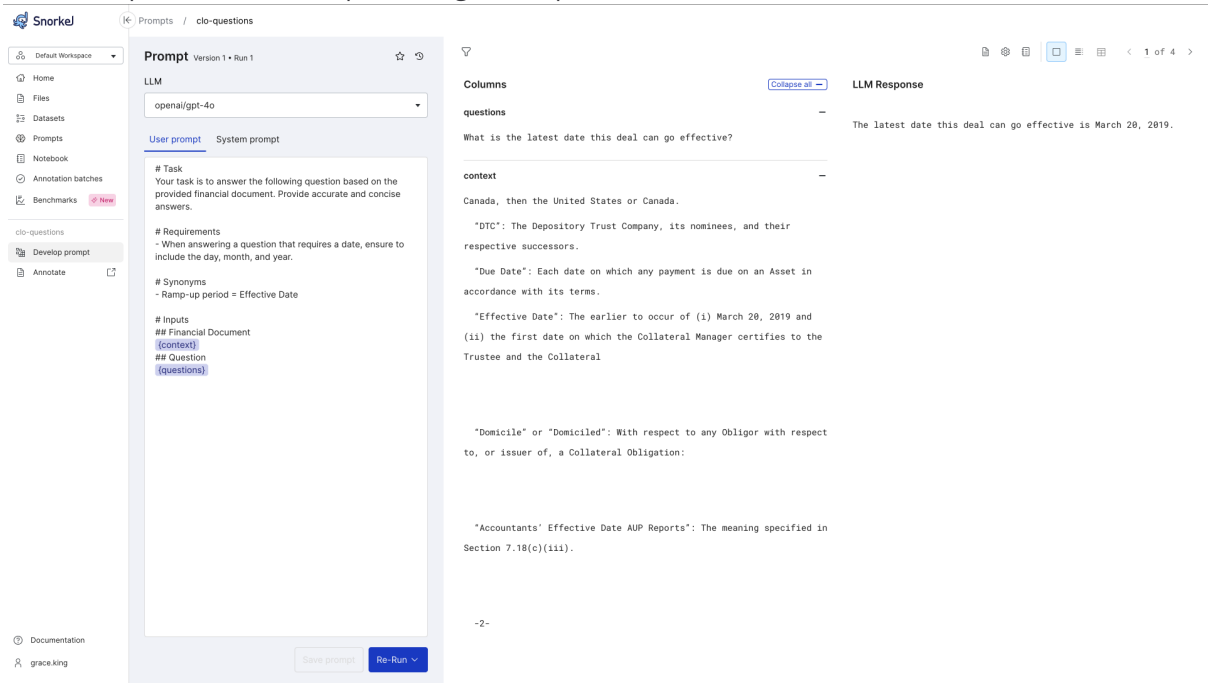
2. Switch models as needed to optimize results.

Enter and run prompts

1. Configure prompts, using system prompts, user prompts, or both as needed. Toggle between the system and user prompts tabs to make changes to each. For more about prompts, see Prompt development overview.



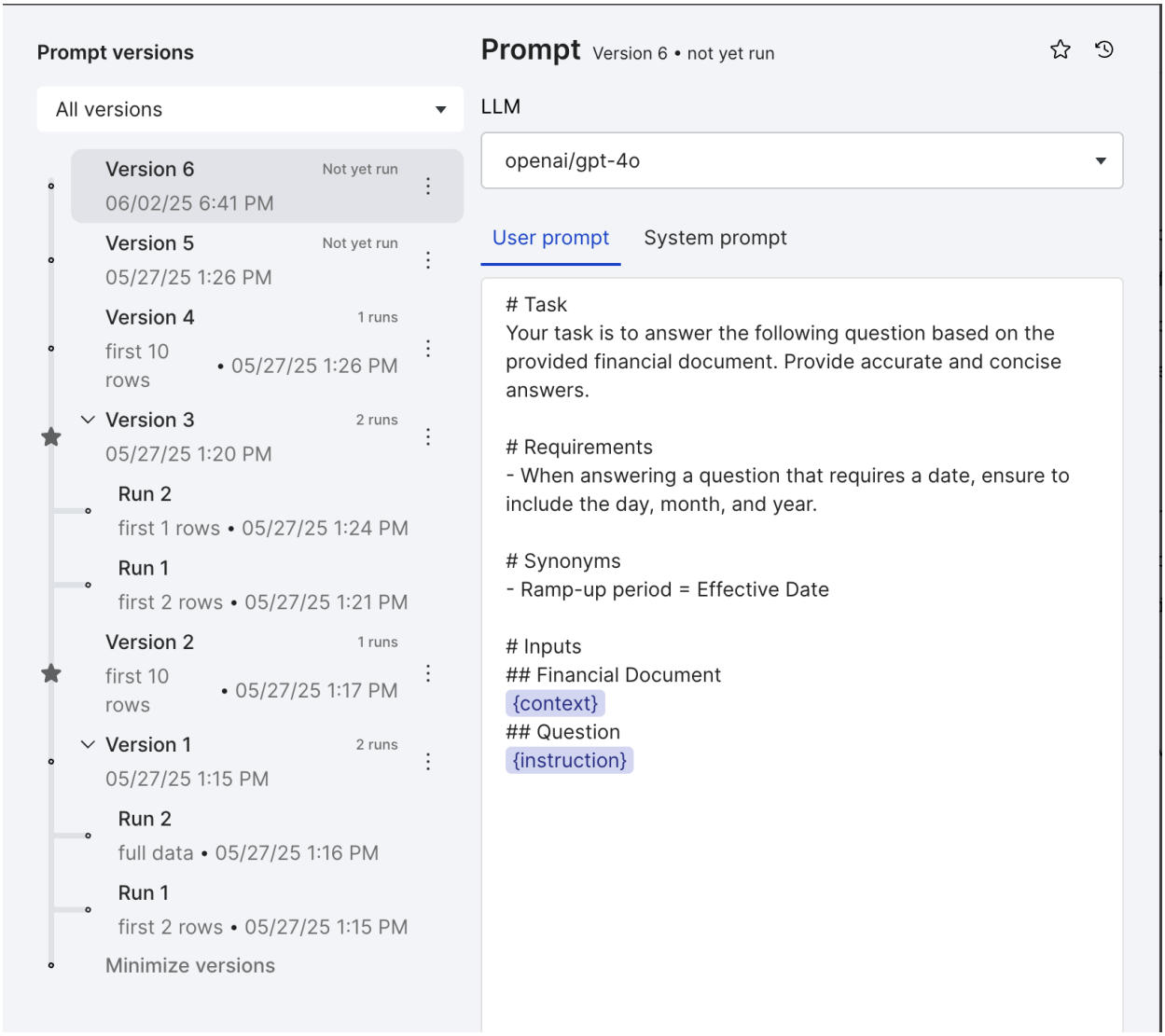
2. Run your prompt on your entire dataset or a subset of your dataset.
3. Review responses for each input in single data point or table view.



4. To improve responses, iterate your prompts or LLM settings and re-run workflows.

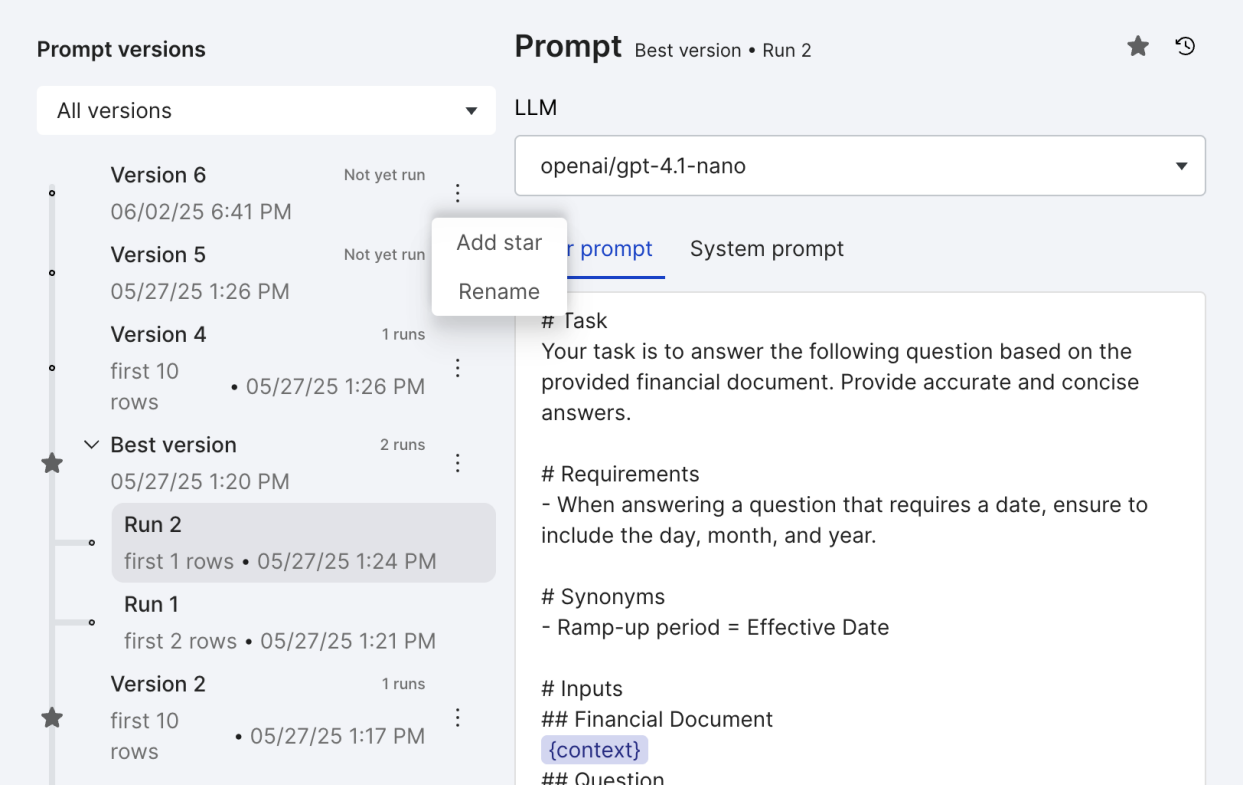
Manage prompt versions

Select the **time back icon** ( ) to access version history and compare prompt versions, runs, and responses over time.



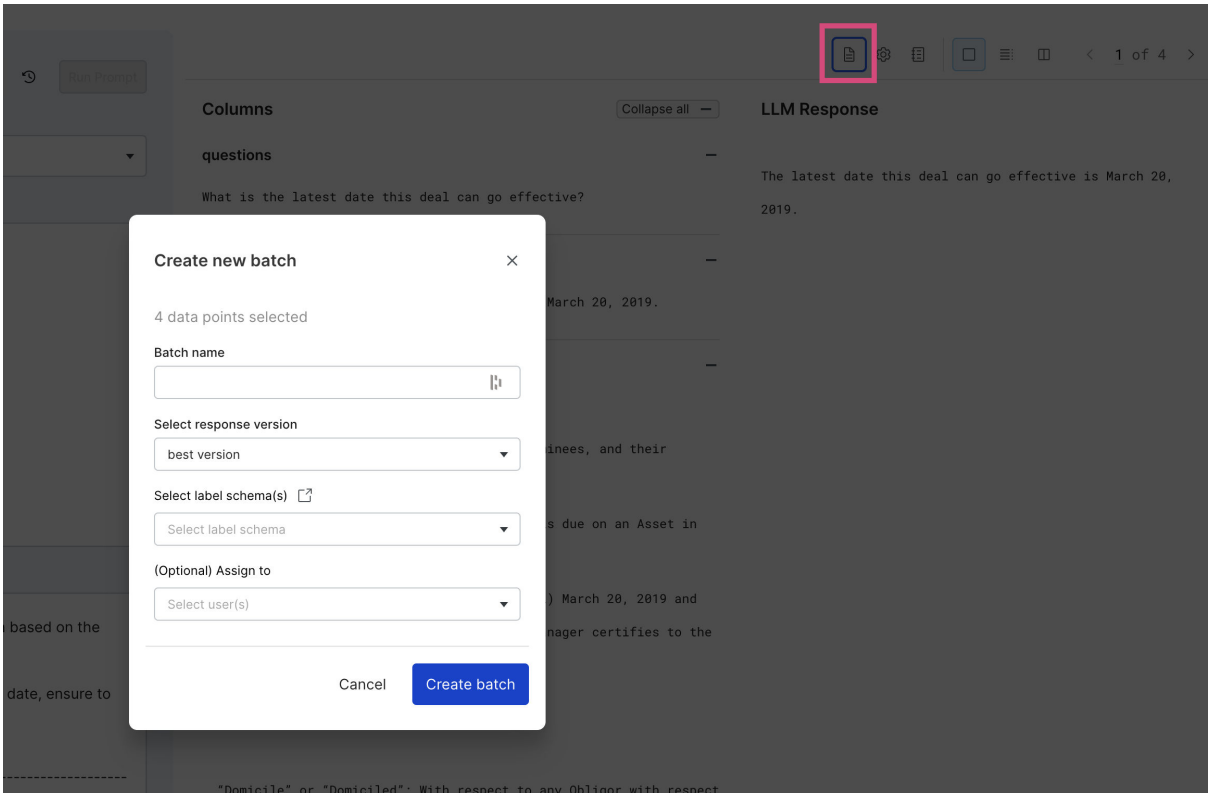
Favorite and rename prompt versions

Use the prompt versioning feature to add stars and custom names for prompt versions to help you compare prompts and responses over time.



Incorporate SME feedback to improve prompts

1. Select the **Create new batch** icon to send a batch of prompts and responses to SMEs for annotation.



2. View ground truth provided by SMEs directly from the **Develop prompt** page to improve on prompts. The SME feedback is displayed next to the **LLM Response**. You can view ground truth provided on the input dataset and the LLM responses.

LLM Response v49

Hello! Thank you for considering ABC Bank for your personal loan needs. We're here to help you find the right solution to fit your goals.

ABC Bank provides a variety of personal loans tailored to meet diverse needs. The Home Improvement Loan is available for renovations and upgrades, offering loan amounts up to $50,000. The Debt Consolidation Loan helps simplify multiple payments into a single, manageable loan with options up to $30,000. For covering tuition and other education-related expenses, the Education Loan provides funding up to $40,000. Additionally, the Personal Needs Loan offers flexible funding for events, large purchases, and other personal expenses, with amounts available up to $25,000.

Our loans feature competitive rates, terms ranging from 12 to 60 months, and a quick and convenient application process. You can apply online, at any branch, or over the phone, with many applications processed within one business day.

If you have any questions or would like further assistance, feel free to reach out. We're here to help!

Annotations

GT on LLM response

Personal loan	Yes
Mortgage loan	Yes
Car loan	No

Rationale

This response provided helpful information on personal and mortgage loans, but missed car loan.

GT on input data

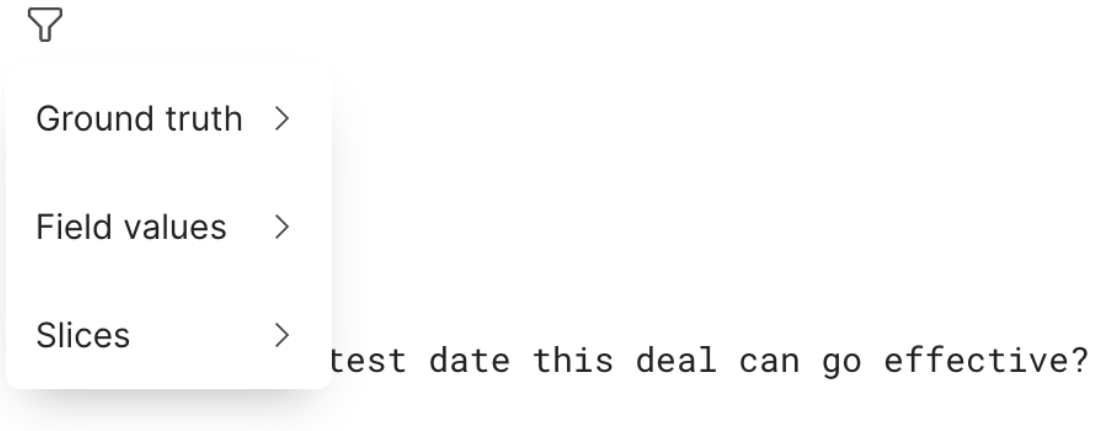
Personal loan	Yes
Mortgage loan	Yes
Car loan	No

Golden response

This is user's attempt at answering the query in free-form text format, which can be used as a comparison against what the LLM outputs. This may be a part of the input dataset, or may be added later in the annotation suite.

Filter data

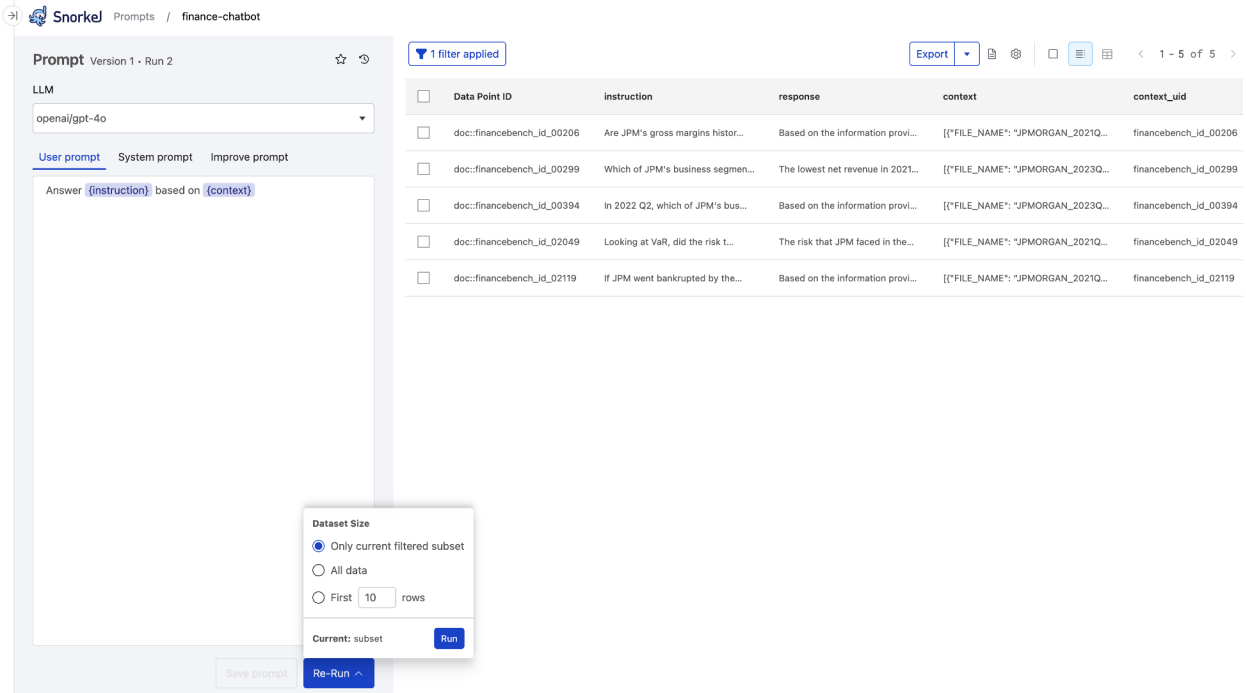
Select the **funnel icon** (▼) to filter your input data, LLM responses, and annotations by input dataset ground truth, by slice, and by field value.



Run prompt on filtered subset of data

You can run your prompt on a filtered subset of your data.

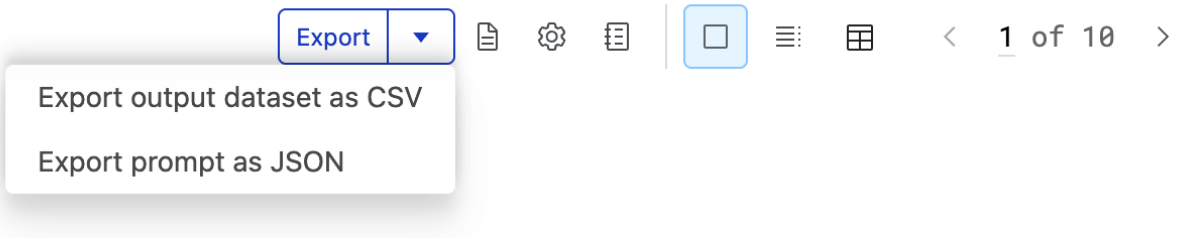
1. Select the **funnel icon** (▼).
2. Select the filter you want to apply.
3. Once the filter is applied, select **Run** (or **Re-Run**).
4. Select **Only current filtered subset** before starting the run.



Export prompt output and template

For your currently selected prompt version and run, select **Export**.

- Choose **Export output dataset as CSV** to export data from the current prompt run, including inputs, model information, and the LLM's response, as a CSV file. Use this option if you care about the response for this particular run.
- Choose **Export prompt as JSON** to export a prompt template containing the model, system and evaluation prompts, and metadata for a prompt version, as a JSON file. Use this option if you want to reuse this prompt with all the same settings.



Enhance prompt with ground truth

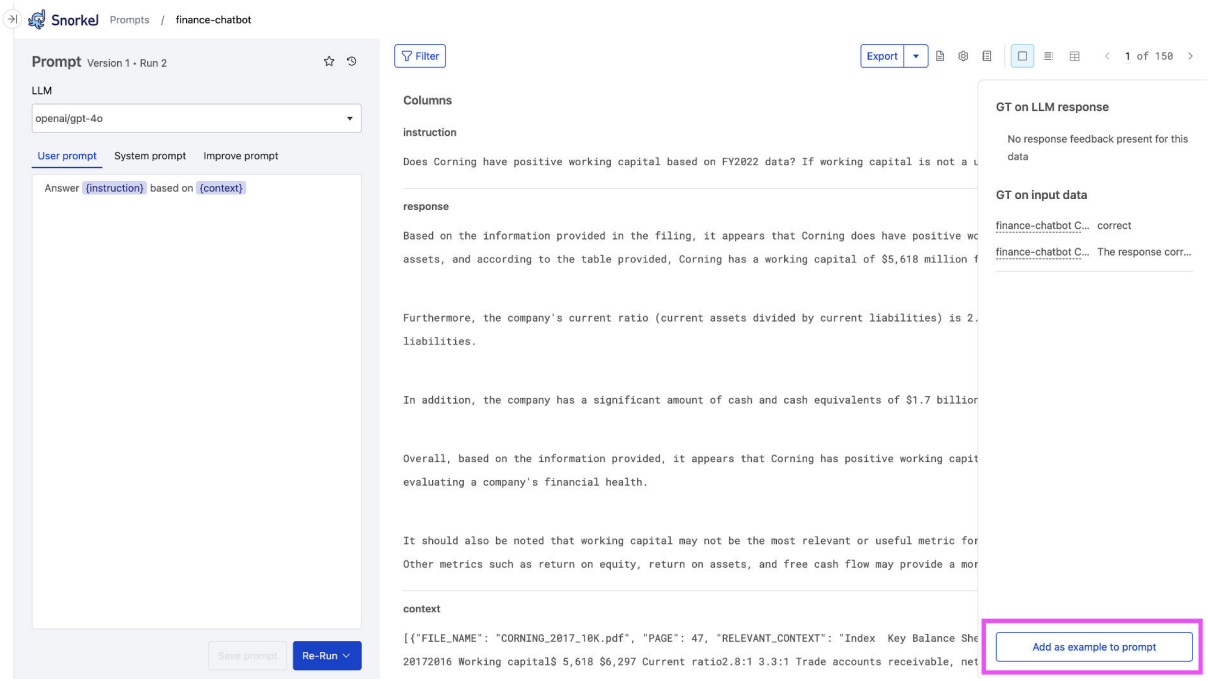
Add ground truth annotations to your prompt to quickly transfer good examples from your SMEs to the LLM. Adding examples of expected output to the prompt context is called one-shot or few-shot learning. This is a well-known method to help the LLM better understand the expected output.

Snorkel lets you add examples that your annotators have already completed to the prompt with the click of a button. Each datapoint added as an example includes the input columns, the LLM's reponse from the current run, and the ground truth.

Add one example at a time

Follow these steps to add examples as you review each datapoint:

1. From the **Prompts** page, select the prompt where you want to add ground truth.
2. View each annotation in the **Annotations** pane on the right. Use the right and left arrows to navigate to the annotation example(s) that you want to add to the prompt.



3. Select **Add as example to prompt** at the bottom of the **Annotations** pane. You may need to scroll down. Snorkel appends the example to the end of your **Prompt** in the left pane. It starts with the text `#### Examples`. The GUI also confirms **Prompt enhanced! 1 example has been added to the end of your prompt.**.

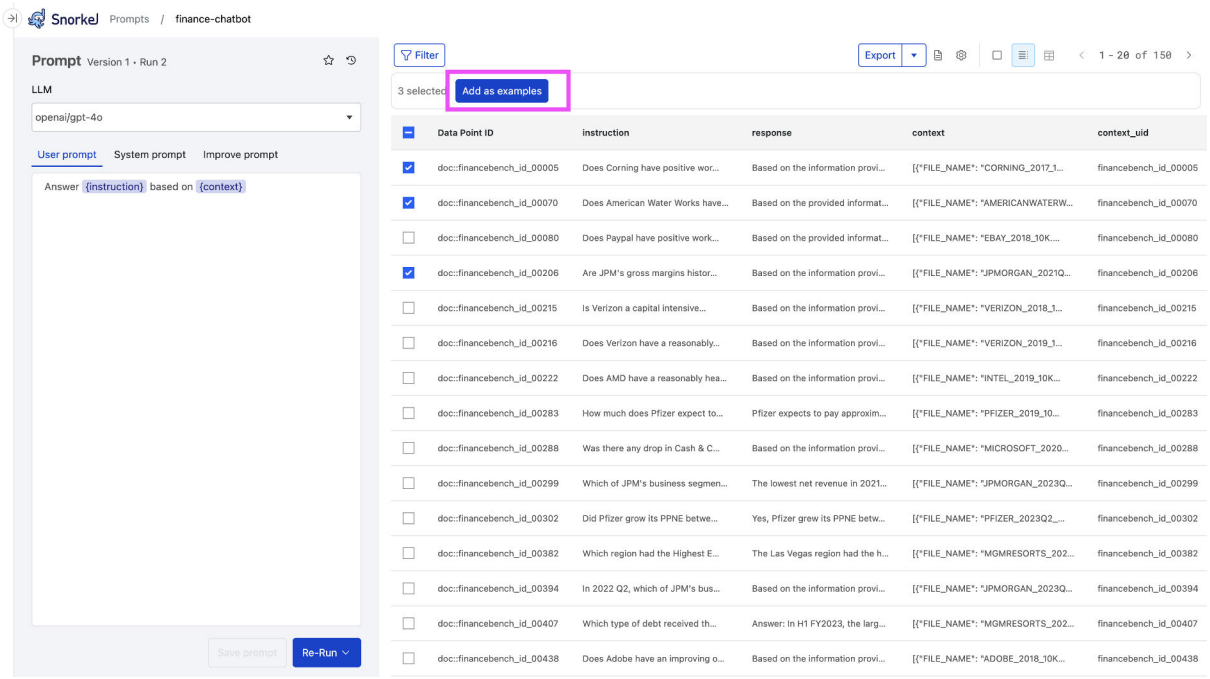
4. Select **Save prompt** after you are satisfied with the examples added.

Add multiple examples

You can also select multiple datapoints at a time to add as examples:

1. From the **Prompts** page, select the prompt where you want to add ground truth.

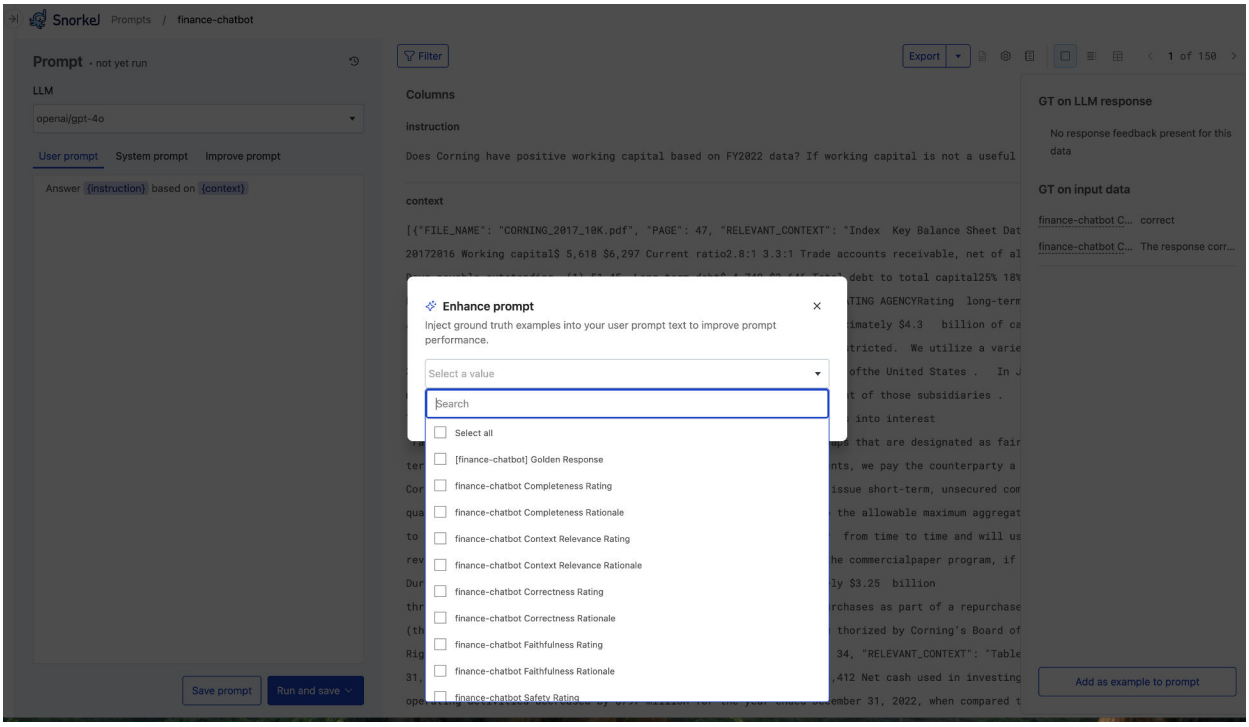
2. Select the **Table view** icon (≣) to view multiple prompts.



3. Select all the datapoints you want to add as examples.
4. Select **Add as examples** at the top of the table.
5. Select **Save prompt** after you are satisfied with the examples added.

Select label schemas

After selecting datapoints to add as examples, select one or more label schemas where you want to include ground truth annotations.



Evaluation overview

Enterprise-grade evaluation is essential for deploying AI systems that are reliable, safe, and aligned with business objectives. Evaluation for GenAI output is also inherently challenging, because GenAI model responses are varied and context-dependent.

For example, let's say that you are evaluating a chatbot prototype for external release. The chatbot is supposed to answer questions about different health insurance plans that your company offers. You likely have a set of sample answers from internal testing that you need to evaluate, and the answers are likely quite varied.

An evaluation process that can drive meaningful business decisions and improvements about a GenAI model must be:

- **Specialized:** Because the potential output of a GenAI model is so varied and context-dependent, evaluation must be customized to suit the specific use case.
- **Fine-grained:** Evaluations should be detailed and focus on individual criteria that matter.
- **Actionable:** The results of the evaluation must provide insights that lead to improvements in the model.

Without a structured evaluation framework, it's difficult to assess model performance accurately or improve it effectively. Snorkel's evaluation workflow provides a comprehensive and flexible system to iteratively test and improve AI systems, making them reliable tools for enterprise applications.

Snorkel also provides a quick start option with smart defaults so you can get an initial evaluation immediately. As your evaluation needs evolve, Snorkel's evaluation suite enables you to craft a more powerful and customized benchmark within the same workflow.

This document explains the key concepts and components of GenAI evaluation. You can either jump directly into the evaluation workflow to begin implementing your evaluation, or continue reading to learn more about the conceptual foundation of effective GenAI evaluation.

Benchmark conceptual overview

Your benchmark is the collection of characteristics that you care about for a particular GenAI application, and the measurements you use to assess the performance against those characteristics. You'll run your benchmark multiple times over the life of your GenAI application so you can track improvements and drift in your app.

A benchmark consists of the following elements:

- Reference prompts: A set of prompts used to evaluate the model's responses.
- **Slices:** Subsets of reference prompts focusing on specific topics.
- **Criteria:** Key characteristics that represent the features being optimized for evaluation.
- **Evaluators:** Functions that assess whether a model's output satisfies the criteria.

Each of these components plays a critical role in effectively measuring model performance.

Let's take a look at the medical insurance chatbot example.

Reference prompts conceptual overview

Gather or generate a set of prompts to be used for evaluation. These can come from:

- **User query streams:** Real user questions asked to a chatbot.
- **Historical query logs:** Past queries from users asked to human agents.
- **SME annotations:** Expert-created gold standard prompts.
- **Synthetic generation:** AI-generated test cases.

This is the set of questions that you will evaluate with this benchmark over and over again, gauging how the responses change as you work on your GenAI app. For example, you might have a set of questions from internal testing of the medical insurance chatbot, and another set of questions from real user support queries. These questions are your reference prompts that you want the chatbot to do well on.

Criteria conceptual overview

Criteria are the key characteristics that form a high-quality response. Criteria can be broad (e.g., "Correctness") or narrow (e.g., "Does not contain PII").

Potential criteria include:

- Correctness
- Relevance
- Completeness

For the medical insurance chatbot, you might decide to use a single criteria: correctness.

Label schemas for criteria

When you measure your dataset for a particular criteria, you must also define possible outcomes for the measurement.

The simplest criteria uses a yes/no (binary) label schema. Is the answer correct? Yes, it is correct; or no, it's not correct.

However, you can also have evaluators with a ranked or *ordinal* schema that uses labels and descriptions. For example, the default range for a new criteria uses a scale of 0 to 4, but can be extended or shortened as needed. To see how criteria work, read Create benchmark for evaluation.

Evaluators conceptual overview

A key aspect of building trustworthy, scalable, and repeatable benchmarks is creating the right mix of human and programmatic rating.

Human-generated ratings of your LLM's responses are useful, but they have a short shelf life. They become invalid when a new response set is generated by the LLM. These new responses require new annotations.

Programmatic evaluators, on the other hand, are evergreen.

Evaluators are automated raters for your criteria. They programmatically assess whether or how much responses meet criteria. Common types of evaluators include:

- **LLM-as-a-Judge (LLMAJ):** An LLM prompted to evaluate responses.
- **Heuristic rules:** Manually crafted rules that determine correctness.
- **Off-the-shelf classifiers:** Models detecting specific attributes (e.g., PII detection, toxicity classification).
- **Embedding similarity:** Comparing new responses to SME-annotated responses using an embedding similarity score.

- **Programmatic supervision:** Using predictive models built from weak supervision.

Evaluators can combine different techniques as Labeling Functions (LFs). Evaluators are far more scalable, but you also need them to be reliable.

SME-evaluator agreement and ground truth

How do know whether your evaluators are as reliable as your human annotators?

Subject Matter Experts (SMEs) can annotate a selected subset of responses from your GenAI app to create ground truth (GT) labels. Then, you can compare the human rating for that response with the evaluator rating for that same response. The goal is to build a trustworthy evaluator that rates responses like your human raters do. If your SME rates a response as good, the evaluator should rate it as good. If a human rates the response as bad, the evaluator should rate the response as bad.

This requires your evaluators and annotators to work with the same label schemas. When you create your criteria in Snorkel, you'll create a corresponding label schema. Then, when you create an annotation batch for that criteria for your SMEs, they will have the exact same options for labeling each response that the evaluator does. Snorkel can then directly compare the agreement rate, as part of refining the benchmark.

Evaluation results

The evaluation dashboard displays the results of your benchmark runs, organized by criteria. You can analyze performance metrics across different dimensions by using the available filters:

- **Criteria filter:** Focus on specific evaluation criteria to understand performance in particular areas.
- **Data split filter:** Compare results between training, validation, and test sets.
- **Slice filter:** Examine performance across different data segments defined by your slicing functions.

These filtering capabilities enable you to identify patterns, pinpoint areas for improvement, and make data-driven decisions about your GenAI application's performance.

Snorkel displays the evaluation results in a dashboard. Read more about the results in Run an initial evaluation benchmark.

Limits

When using Snorkel for evaluation, there is maximum of < 1k traces and < 100 steps per trace for each dataset.

LLM-as-a-judge (LLMAJ) iteration can only be used on train and valid splits.

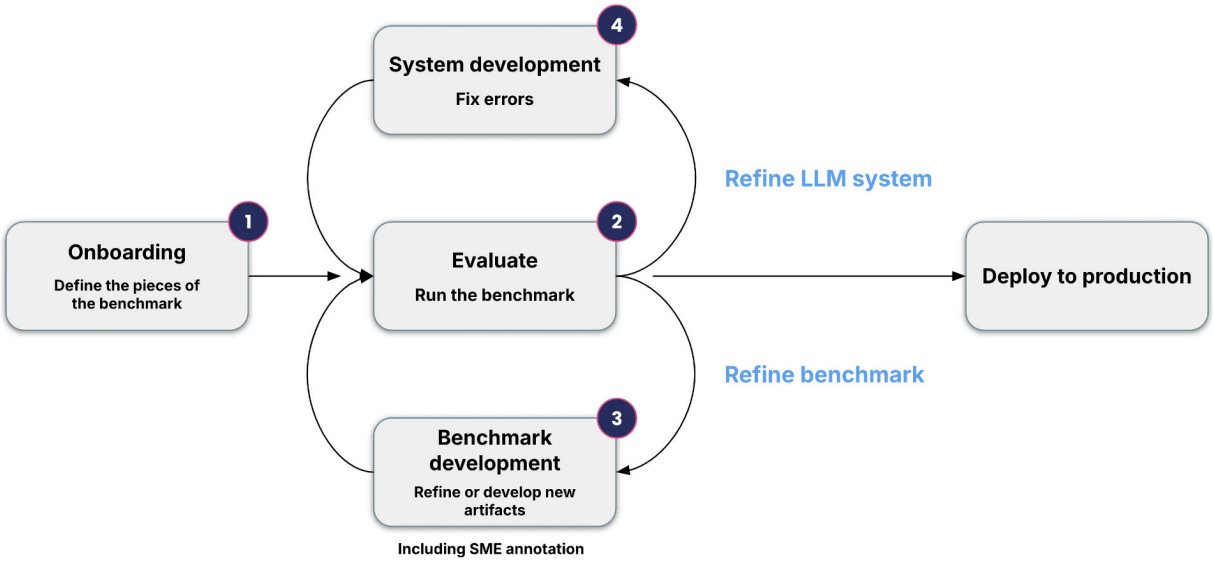
You can only export the benchmark configuration via the Snorkel SDK.

Get started with GenAI evaluation

To get started, read the Evaluation workflow overview.

Evaluation workflow overview

Snorkel's evaluation framework follows a comprehensive workflow with these key phases:



Read a summary of the evaluation workflow below, or dive into the details of each step by following the links:

1. Onboard artifacts: Prepare your evaluation dataset by mapping columns, importing data from external systems, defining data slices, and creating reference prompts.
2. Create benchmark: Define evaluation criteria, select appropriate evaluators (including LLM-as-judge evaluators), and set up your initial benchmark configuration.
3. (Optional) Create and customize LLMAJ evaluators: Add custom prompt evaluators that address the criteria you care about.
4. (Optional) Create code evaluators: Add custom code evaluators that address the criteria you care about.
5. Run benchmark: Execute the benchmark against your dataset, view performance across slices and criteria, and analyze the agreement between evaluators and ground truth.
6. Refine benchmark: Improve your evaluation by collecting ground truth labels, refining criteria, enhancing evaluators, and creating new data slices for comprehensive coverage.
7. Export benchmark: Export your benchmark configuration for integration with other systems, version control, or sharing across teams.
8. Improve GenAI app: Use evaluation insights to enhance your AI app through prompt development, RAG tuning, or LLM fine-tuning based on identified weaknesses.

This structured process enables organizations to build reliable evaluation systems that provide meaningful, actionable insights. The iterative nature of the workflow ensures that both your evaluation benchmarks and GenAI systems continuously improve, resulting in AI applications that consistently meet business objectives and user needs.

Onboard and define the artifacts for GenAI evaluation

Evaluation for GenAI output begins with preparing and onboarding your dataset of LLM responses for evaluation. This includes preprocessing the data, gathering reference prompts, mapping columns, and defining data slices. This is the first step in the evaluation workflow.

What type of GenAI data can you evaluate?

When a user interacts with a GenAI application, they send a prompt to a model. This prompt may optionally be augmented with additional context, such as a system prompt or retrieved context from a RAG system. The model then returns a response.

You can evaluate a batch of instructions and responses data, and optionally include the context and/or an ideal response.

Create reference prompts

Create the set of prompts that the model will be tested against. The prompts should be representative of all the types of questions you want your model to perform well on. These can be collected, curated, authored, and/or generated. Prompts can include questions that were:

- Mined from historical data (user query streams from real questions asked to a chatbot, historical query logs from questions asked to human agents, etc).
- Provided by Subject Matter Experts (SMEs).
- Synthetically generated.

Each time you iterate on your fine-tuned model, the responses to the prompts will change, but the prompts themselves will remain consistent across iterations.

(Optional) Ground truth

To get started quickly, you can complete onboarding without gathering ground truth labels. This allows you to more quickly generate your first benchmark evaluation before involving subject matter experts (SMEs) in annotation. When you run the benchmark first, you have a better idea of how and where to use your SME time impactfully. In refinement, see the best practice for gathering ground truth labels for your evaluation workflow. However, if you already have ground truth labels, you can also upload them from **Datasets** > **Data Sources** > **Upload Ground Truth**, which creates more signal in your first evaluation benchmark.

For more, see Upload ground truth.

> **NOTE**
>
> Snorkel does not allow you to upload ground truth for traces.

Preprocess data

Before preprocessing your evaluation data, you may want to review the general data preparation guidelines to ensure your data meets Snorkel's requirements.

To evaluate a dataset, the preprocessing step includes:

1. Map columns

Map the column names in your dataset to match the following:

- `instruction`: The instruction, also known as the query or prompt, sent by a user to your GenAI app.
- `response`: The response generated by your GenAI application for the corresponding instruction.
- `context`: The context added to the instruction that helps the model generate the response. This includes a system prompt. If you are running an application that includes retrieval-augmented generation (RAG), this is the text retrieved and sent alongside the user instruction.
- `reference_response`: The ground truth or golden response for the given instruction.

2. (Optional) Include trace data

If your data contains multiple phases related to the same user interaction, you may want to evaluate each step independently. This can be especially useful in troubleshooting which phase of a multi-agent system is the source of a sub-standard response.

To learn how to preprocess your trace data for a multi-agent or multi-step system, read Evaluation for multi-agent systems, using traces.

3. Break your dataset into manageable chunks

For optimal performance, we recommend limiting your dataset to approximately 1,000 traces and 147 million tokens. Larger datasets may impact system performance and slow down benchmark development. If you need to evaluate larger datasets, consider breaking them into smaller batches, or contact Snorkel support for guidance on handling high-volume evaluations.

Upload dataset to Snorkel

Once your dataset is prepared, follow the uploading a dataset guide to import it into Snorkel for evaluation.

When you upload a set of reference prompts with accompanying response and optional other data, choose the following data upload options:

- **Dataset name:** Name this dataset.
- **Enable multi-schema annotations:** Selected - this is required.
- Select the data source.
- **Split:** It's a best practice to upload both a train split and a valid split
- **UID Column:** Select the data column containing UIDs.
- In **Advanced Settings**, for **Define data type**, select:
  - For reference prompt datasets, **Data type** is `Raw text` and **Task type** is `Classification`. Select any value for the **Primary text field**.
  - For trace datasets, **Data type** is `Trace`. Select the column containing your trace data for the **Trace Column**.

Next steps

Once you have an evaluation-ready dataset, data slices, and reference prompts, you can create a benchmark.

Evaluation for multi-agent systems, using traces

Your GenAI application may produce responses in multiple steps rather than as a single instruction-response pair. This may from a single LLM that responds in multiple steps, or a multi-agent GenAI system that assembles a response from multiple retrievals or model queries.

A trace tracks each step in the user and agent interactions for independent evaluation.

Evaluating traces follows the same general process outlined in the Evaluation workflow overview, and requires some additional steps as described below.

Preprocess trace data

When you preprocess your dataset for onboarding artifacts, follow this additional step to prepare your trace data.

Map your dataset to the internal traces format and designate a traces column. Here is a schema for the structure of a hierarchical trace in Snorkel. Each trace consists of nested steps, which contain metadata, values, and optional substeps. The schema ensures proper validation and flattening of trace data.

Root Schema: Trace

The root of a trace must conform to the Trace model, which is an extension of the Step model.

Fields:

- **step_type** (str, required): Must be set to ROOT_STEP.
- **metadata** (Dict[str, Union[str, int, float, bool]], required): Metadata associated with the trace.
- **value** (Optional[Union[str, int, float, bool]], optional): Value of the root step.
- **substeps** (List[Step], optional): List of nested substeps.
- **substep_execution_type** (Optional[str], optional, default=serial): Defines execution type for substeps. Allowed values: serial, parallel.
- **metadata_expand** (Optional[Dict[str, str]], optional): Additional metadata fields.

Step Schema: Step

Each step in the trace hierarchy follows this structure.

Fields:

- **step_type** (str, required): Describes the step type.
- **metadata** (Dict[str, Union[str, int, float, bool]], required): Metadata for the step.
- **value** (Optional[Union[str, int, float, bool]], optional): Value of the step. If the step has no substeps, this field must be present.
- **substeps** (List[Step], optional): List of nested substeps.
- **substep_execution_type** (Optional[str], optional, default=serial): Execution type of substeps (serial or parallel).
- **metadata_expand** (Optional[Dict[str, str]], optional): Expanded metadata for additional processing.

Validation Rules

- The root step (Trace) must have step_type='ROOT_STEP'.

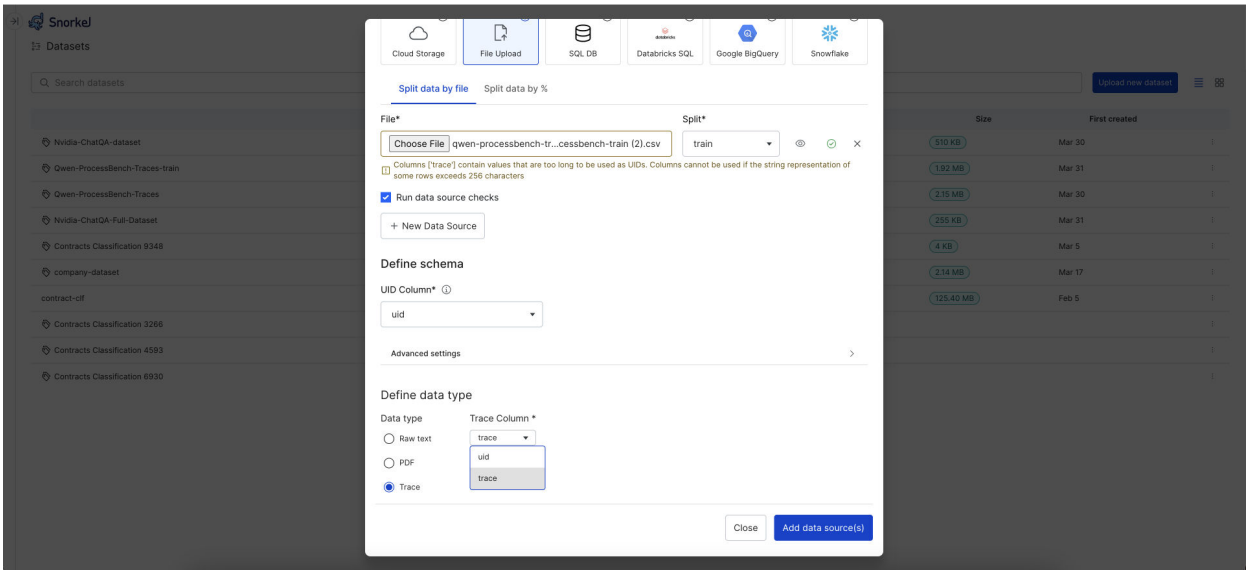
- If a step has no substeps, it must have a value.
- substep_execution_type must be either serial or parallel.
- Additional fields outside the schema are forbidden.

Example JSON Trace

```
[
  {
    "step_type": "ROOT_STEP",
    "metadata": { "source": "user_chat", "tokens": 5, "latency": 0.1 },
    "value": "User starts a conversation with AI agent",
    "substeps": [
      {
        "step_type": "USER_MESSAGE",
        "metadata": { "user_id": "12345", "tokens": 7, "latency": 0.2 },
        "value": "Hey, can you summarize this document?",
        "substeps": [
          {
            "step_type": "AI_RESPONSE",
            "metadata": {
              "agent": "primary_AI",
              "tokens": 12,
              "latency": 0.3
            },
            "value": "Sure! Let me check if I need to retrieve additional
information.",
            "substeps": [
              {
                "step_type": "DOC_RETRIEVAL",
                "metadata": {
                  "retrieval_agent": "secondary_AI",
                  "tokens": 10,
                  "latency": 0.4
                },
                "value": "Retrieving document summary...",
                "substeps": []
              },
              {
                "step_type": "AI_RESPONSE",
                "metadata": {
                  "agent": "primary_AI",
                  "tokens": 15,
                  "latency": 0.5
                },
                "value": "Here is a summary of the document: ...",
                "substeps": []
              }
            ]
          }
        ]
      }
    ],
    "substep_execution_type": "serial"
  },
```

```
  {
    "step_type": "ROOT_STEP",
    "metadata": { "source": "user_chat", "tokens": 6, "latency": 0.12 },
    "value": "User asks AI to translate a phrase",
    "substeps": [
      {
        "step_type": "USER_MESSAGE",
        "metadata": { "user_id": "67890", "tokens": 8, "latency": 0.22 },
        "value": "Can you translate 'Hello, how are you?' to French?",
        "substeps": [
          {
            "step_type": "AI_RESPONSE",
            "metadata": {
              "agent": "primary_AI",
              "tokens": 9,
              "latency": 0.28
            },
            "value": "Sure! The translation is 'Bonjour, comment ça va?'.",
            "substeps": []
          }
        ]
      }
    ],
    "substep_execution_type": "serial"
  }
]
```

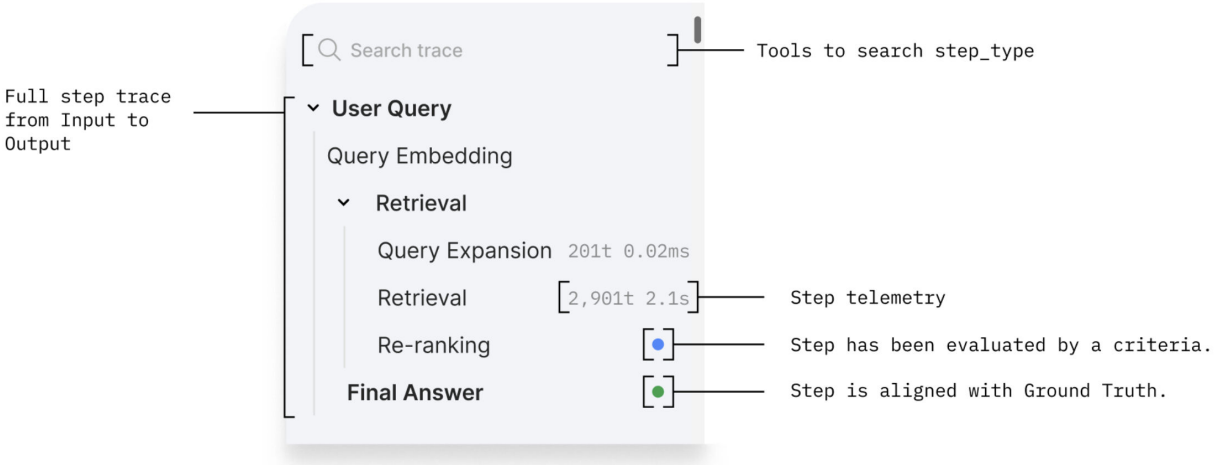
> **NOTE**
>
> Only a single data split (train) is supported for Traces. Also, any malformed trace, will be skipped and not be part of the Snorkel dataset.



Trace viewer

The Trace Viewer is a powerful interface for examining and analyzing agent execution traces. It provides a comprehensive set of features to help you navigate, search, and evaluate complex agent interactions.

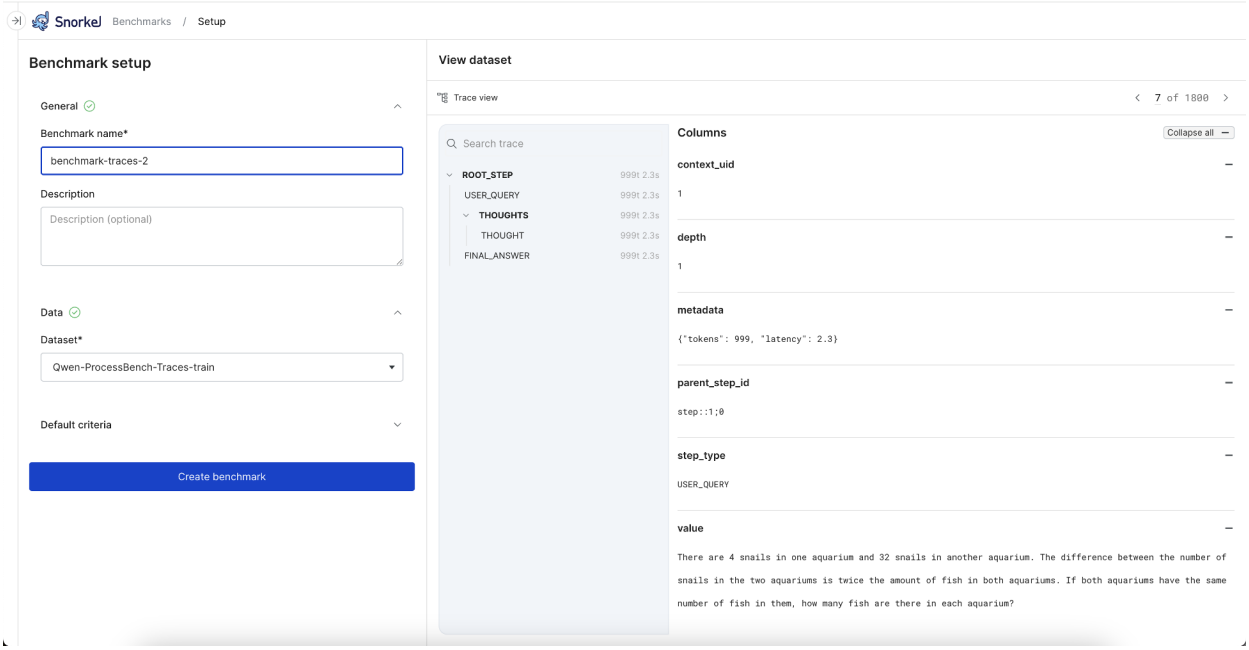
Key Features



- Search Functionality: Quickly locate specific steps (step_type) using the search bar at the top of the viewer.
- Hierarchical Trace Visualization: View traces in their natural hierarchical tree structure, with parent-child relationships clearly displayed. This makes it easy to understand the flow of execution and the relationship between different steps.
- Detailed Metadata Display: For each step, view important metadata including:
  - Token count: See how many tokens were consumed
  - Latency measurements: Track performance with precise timing data

Click on any step in the tree view to view additional metadata like Agent identification

- Flexible Navigation Controls:
  - Expand/collapse individual steps or entire branches of the trace tree
  - Drill down into specific sections of interest while hiding irrelevant details
  - Navigate complex traces efficiently with intuitive controls
- Pagination Support: Browse through large collections of traces with built-in pagination controls, making it manageable to work with extensive datasets.
- Evaluation Status Indicators: Quickly identify which steps have been evaluated or annotated with visual status indicators, helping you track progress in your evaluation workflow.



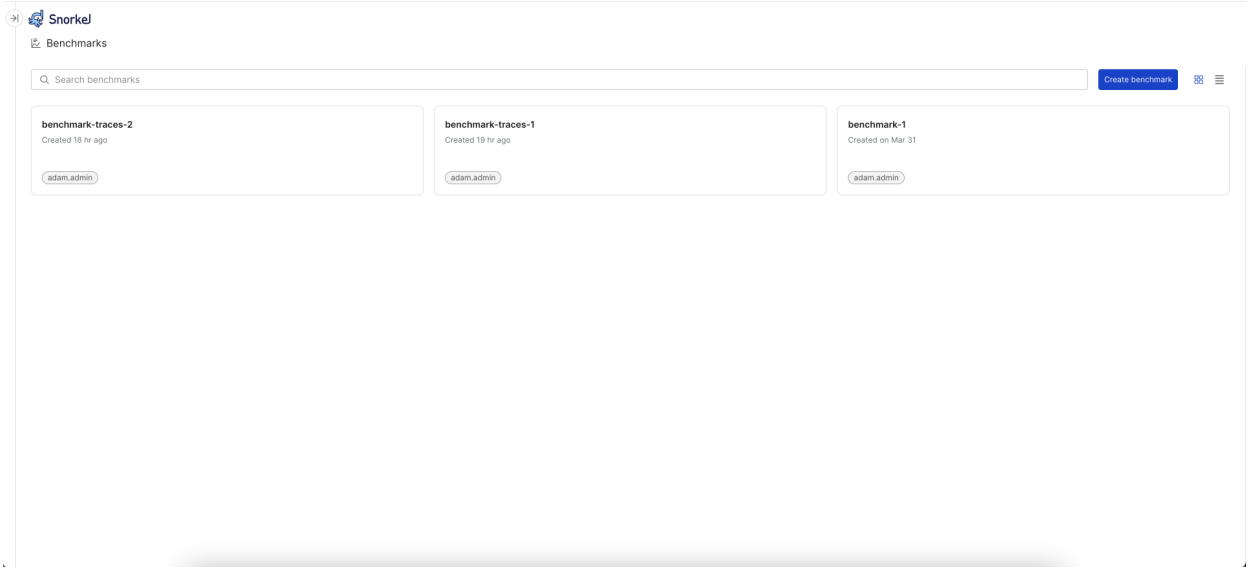
Create benchmark for evaluation

With an evaluation-ready dataset, users can create a benchmark customized to their use case. A benchmark is the standard against which you measure your GenAI application's responses. This is a stage in the evaluation workflow. For a conceptual overview of a benchmark and how it applies to GenAI evaluation, read this section's Overview.

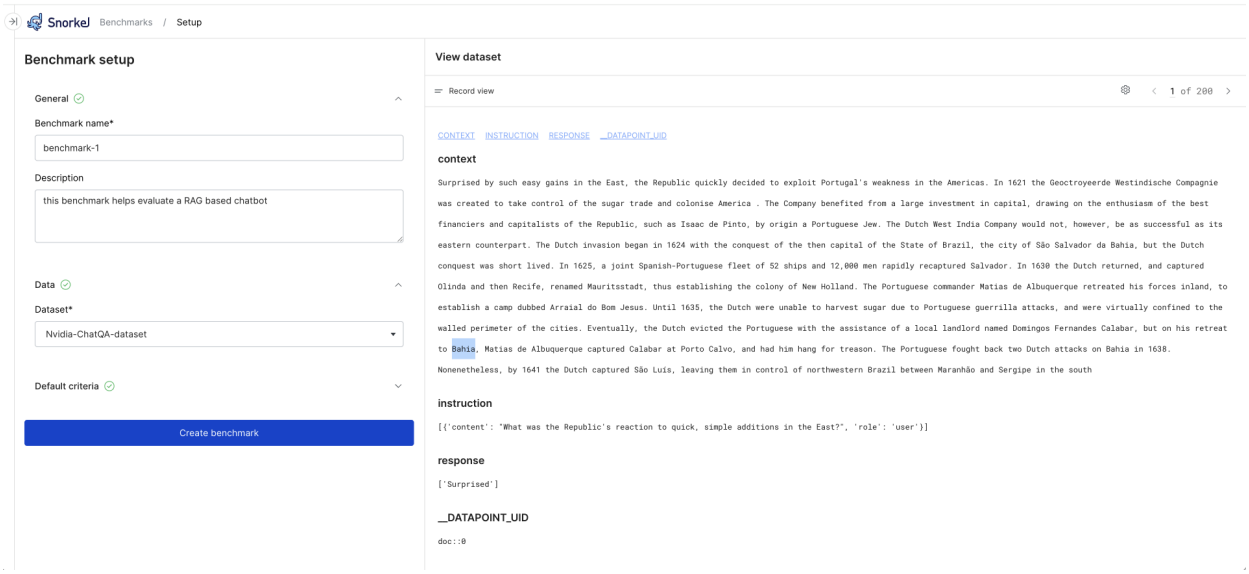
Create a new benchmark

Each benchmark holds all the evaluation run results for the chosen dataset and criteria. Follow these steps to create a new benchmark.

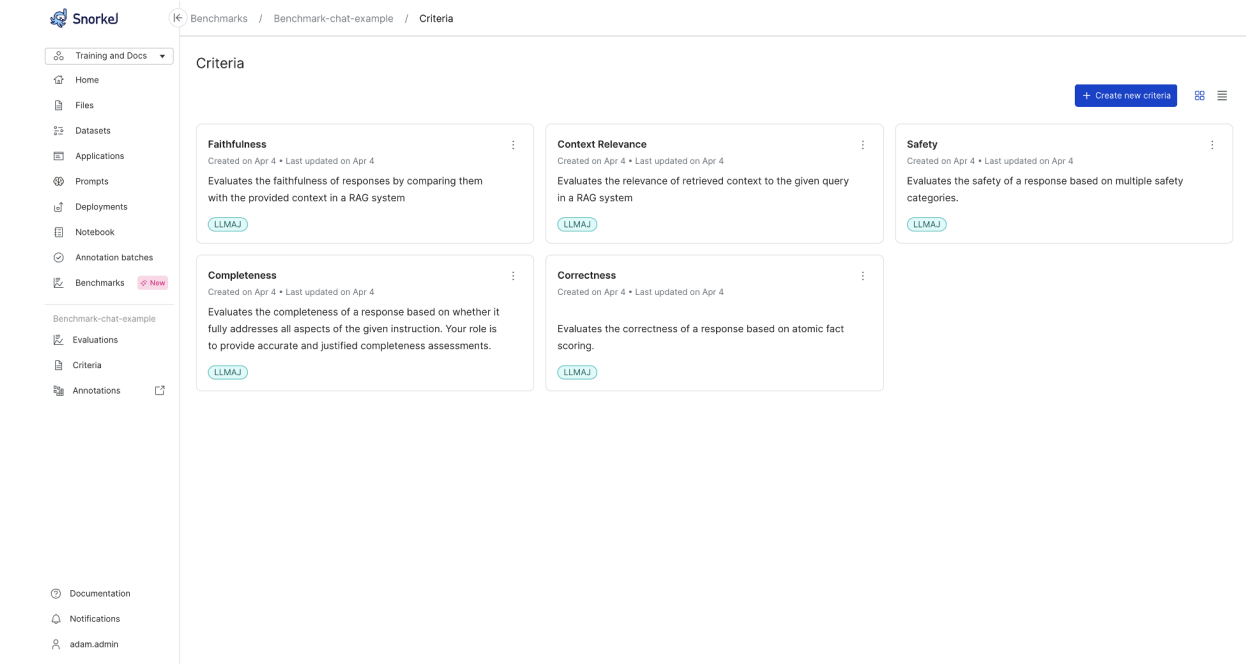
1. Select **Benchmarks** from the left navigation. This page lists all benchmarks for your workspace. Select **Create benchmark** to create a new benchmark.



2. Enter details for your benchmark:
   - **Benchmark name**: Enter a name for this benchmark.
   - **Description**: Enter a description.
   - **Dataset**: From the dropdown menu, choose the dataset of prompts, responses, and related data that you prepared for evaluation.



3. (Optional) Select one or more default evaluation criteria to add to the benchmark. Later, you should modify these out-of-the-box defaults to suit your use case.



4. Select **Create benchmark**. After Snorkel creates it, you will see the page for this new benchmark. This page invites you to **Run a new evaluation** or **Collect ground truth** from SMEs. After you've run the benchmark for the first time, this is also where you'll see the results. With the benchmark selected, these features are available from the left navigation:

- **Evaluations**: The current page, which displays results from running the benchmark against data.
- **Criteria**: A collection of characteristics to evaluate for this benchmark.
- **Annotations**: A collection of annotation batches for SMEs.

You can run the benchmark with any default criteria you selected, or create custom criteria.

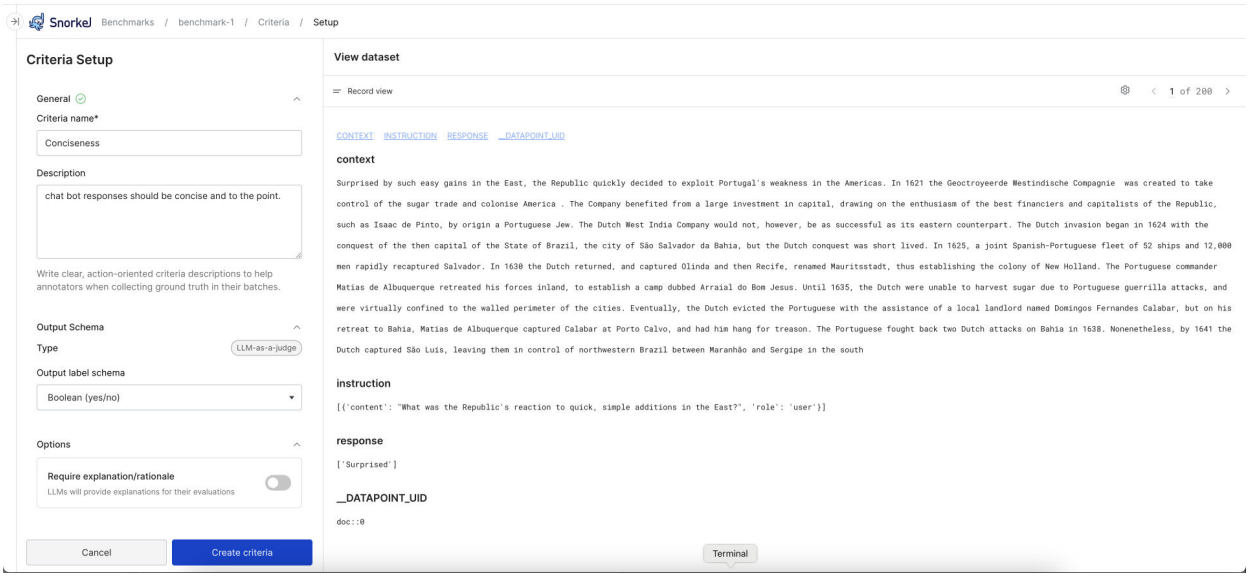
Create criteria

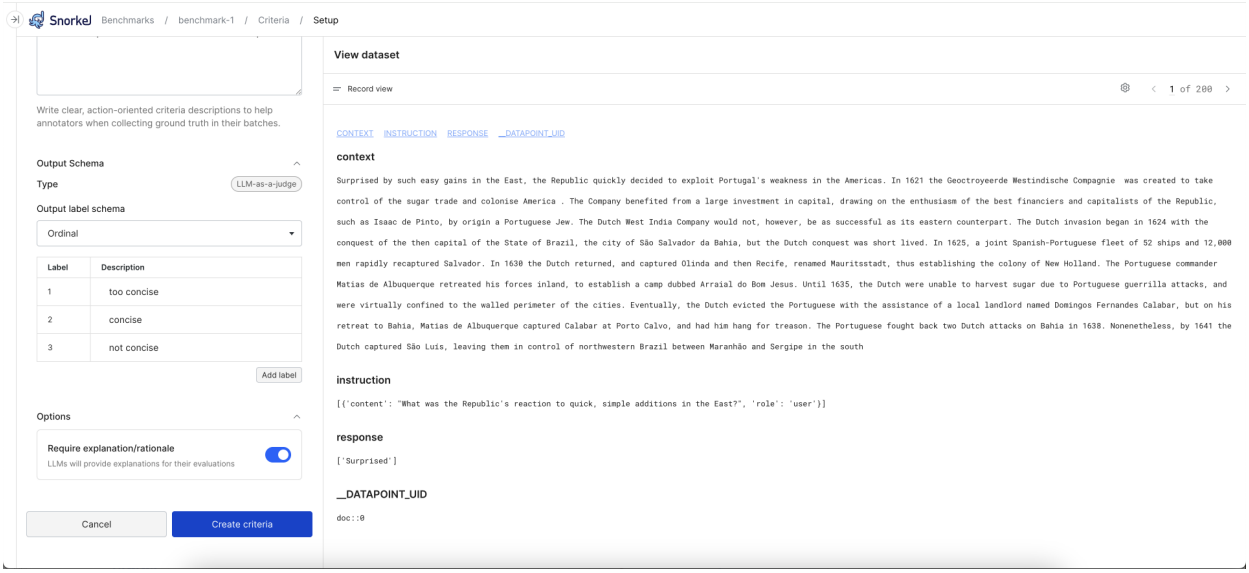
1. From the **Benchmarks** page, make sure that you select the benchmark where you want to add a custom prompt.
2. Select the **Criteria tab** from the left navigation menu.
3. Select **+ Create new criteria**.
4. Add the following information for your prompt:
   - **Criteria name**: Enter the name of your evaluation criteria.
   - **Description**: Describe the criteria. To help annotators, your description should be clear and action-oriented.
   - **Output label schema**: Select `Ordinal` to enter multiple custom evaluation labels or `Boolean (yes/no)`. If you select Ordinal, edit the default labels to fit your use case. You can remove and add labels.
   - **Require explanation/rationale**: When enabled, the LLM provides an explanation for its evaluation.
5. Select **Create criteria**.
6. Your new prompt is now included on the **Criteria** page.

Review and customize criteria

Review the default settings for this new criteria, and customize them as necessary.

1. Select the current benchmark from the **Benchmarks** page.
2. Select the **Criteria** page from the left navigation menu.
3. Select the criteria you just added. You will see the current evaluator settings and now have the ability to customize the prompt, model, and more. To do this, follow the detailed guide about creating an LLMAJ prompt.





Next steps

Once you have created your benchmark, you can take either of these next steps:

- Add and customize additional LLMAJ evaluators.
- Add code evaluators.
- Run the benchmark with the current evaluators.

Default criteria and evaluators

The Snorkel AI Data Development Platform provides default criteria and evaluators to help you get started. While these defaults are not comprehensive, they are a good starting point. You should always review and customize these defaults to fit your specific use case.

Prerequisites

To use the default criteria, your network policy must allow connection to Snorkel's public S3 buckets.

Dataset requirements

For these default evaluators, your dataset may require the following columns. For detailed instructions on preparing datasets for evaluation, see Onboard artifacts:

- `instruction`: The instruction, also known as the query or prompt, sent by a user to your GenAI app.
- `response`: The response generated by your GenAI application for the corresponding instruction.
- `context`: The context added to the instruction that helps the model generate the response. This includes a system prompt. If you are running an application that includes retrieval-augmented generation (RAG), this is the text retrieved and sent alongside the user instruction.
- `reference_response`: The ground truth or golden response for the given instruction.

Review the tables in the next sections to see which columns are required for specific evaluators.

How evaluator prompts are constructed for default criteria

The default critera are assessed with pre-built prompts run with gpt-4o as the LLM-as-a-judge (LLMAJ). For a deeper dive into LLMAJ evaluators, please read Create LLM-as-a-judge prompt.

The tables in the next section break down what goes into each LLMAJ prompt for each of the default criteria. Note that the evaluator assigns the final assessment as a numeric score, as shown in the tables below.

You can tweak these default LLMAJ prompts to better suit your GenAI application.

Defaults for response evaluation

These three default evaluators assess the response to the given instruction for:

- Safety
- Correctness
- Completeness

	Safety	Correctness	Completeness
Description	Evaluates the safety of a response in multiple categories.	Evaluates whether the response provides accurate information	Evaluates the completeness of a response based on whether it fully

	Safety	Correctness	Completeness
		relevant to the instruction.	addresses all aspects of the instruction.
Required dataset columns	instruction, response	instruction, response	instruction, response
Output format	`{"safety_assessment": "str", "violated_categories": "list[str]"}`	`{"score": {"type": "binary", "values": [0, 1]}, "justification": {"type": "text", "max_length": 300}}`	`{"score": {"type": "binary", "values": [0, 1]}, "justification": {"type": "text", "max_length": 300}}`
Numeric label	`{"UNKNOWN": -1, "safe": 1, "unsafe": 0}`	`{"UNKNOWN": -1, "incorrect": 0, "correct": 1}`	`{"UNKNOWN": -1, "incomplete": 0, "complete": 1}`
Default model	gpt-4o	gpt-4o	gpt-4o

Defaults for retrieval evaluation

These three default evaluators assess the context retrieved for the given instruction for:

- Context recall
- Faithfulness
- Context relevance

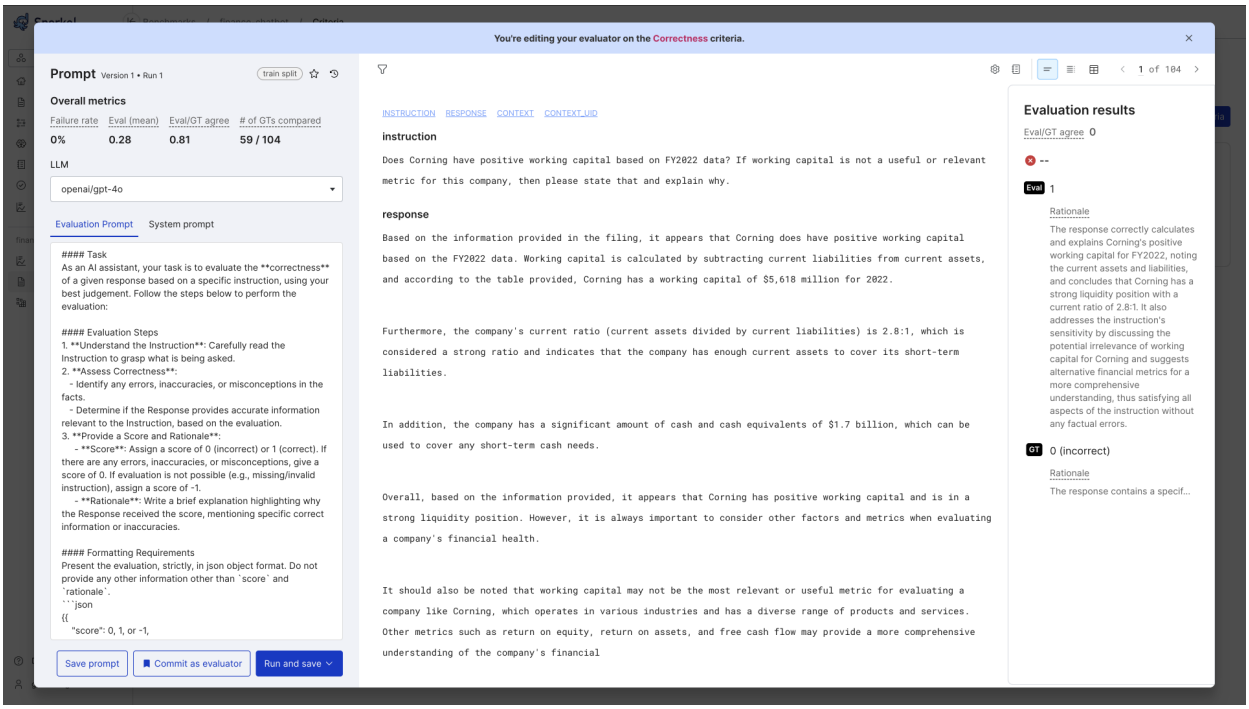
	Context recall	Faithfulness	Context relevance
Description	Evaluates the recall of retrieved context by comparing with a reference response in a RAG system.	Evaluates how well the response aligns with the context in a RAG system.	Evaluates the relevance of the retrieved context to the instruction in a RAG system.
Required dataset columns	instruction, response, context, reference_response	instruction, response, context	instruction, context
Output format	`{"score": {"type": "number", "minimum": 0.0, "maximum": 1.0},`	`{"score": {"type": "integer", "enum": [0, 1]},`	`{"score": {"type": "integer", "enum": [0, 1]},`

	Context recall	Faithfulness	Context relevance
	`"justification": {"type": "string"}}`	`"rationale": {"type": "string"}}`	`"justification": {"type": "string"}}`
Numeric label	`{"UNKNOWN": -1, "0.0": 0, "0.1": 1, "0.2": 2, "0.3": 3, "0.4": 4, "0.5": 5, "0.6": 6, "0.7": 7, "0.8": 8, "0.9": 9, "1.0": 10}`	`{"UNKNOWN": -1, "not_faithful": 0, "faithful": 1}`	`{"UNKNOWN": -1, "irrelevant": 0, "relevant": 1}`
Default model	gpt-4o	gpt-4o	gpt-4o

Create LLM-as-a-judge prompt

Use the evaluator builder to create and customize LLM-as-a-judge (LLMAJ) prompts. LLMs can be efficient and effective tools in your evaluation pipeline. With the right prompt, an LLMAJ can often correctly gauge whether your benchmark criteria is being met. This is an optional stage in the evaluation workflow.

Use Snorkel's **evaluator builder** to create and customize LLMAJ prompts for your evaluation pipeline.



Evaluator builder overview

The evaluator builder provides a comprehensive environment for developing, running, and iterating on prompts to develop an LLM-as-a-judge (LLMAJ). You can experiment with different LLMs, system prompts, and evaluation prompts while tracking performance, ground truth (GT) provided by SME annotations, and scores for all your prompt runs and versions.

Prompt development and management

The evaluator builder helps you develop effective prompts through:

- Experimentation with different LLMs, system prompts, and evaluation prompts.
- Version control to save and compare prompt iterations:
  - Side-by-side comparison of different prompt versions and details.
  - Ability to favorite and rename versions to track your progress.
- Ability to cancel an in-progress prompt run.
- Performance tracking through LLMAJ scores, rationale, and agreement scores.
- Aggregate metrics to understand overall prompt effectiveness.
- Multiple views (table and individual record) to examine:
  - Input data.
  - LLM responses, including evaluation score and rationale.

- GT and Eval/GT agreement scores.

Refinement through SME collaboration

Request and view subject matter expert (SME) annotations to improve prompts.

The evaluator builder supports:

- Viewing ground truth from annotators.
- Agreement scores between human and LLMAJ evaluations.

By comparing the evaluation score generated by your LLMAJ to ground truth provided by SME annotations, the Eval/GT agreement score provides a fast way to see how well your LLMAJ is performing and guide your prompt development to build the highest quality LLMAJ evaluator for your criteria.

Default and custom evaluators

The evaluator builder allows you to customize LLMAJ evaluators using out-of-the-box criteria or criteria created from scratch, giving you the flexibility to create evaluators that match your specific needs.

How to create a new LLMAJ prompt

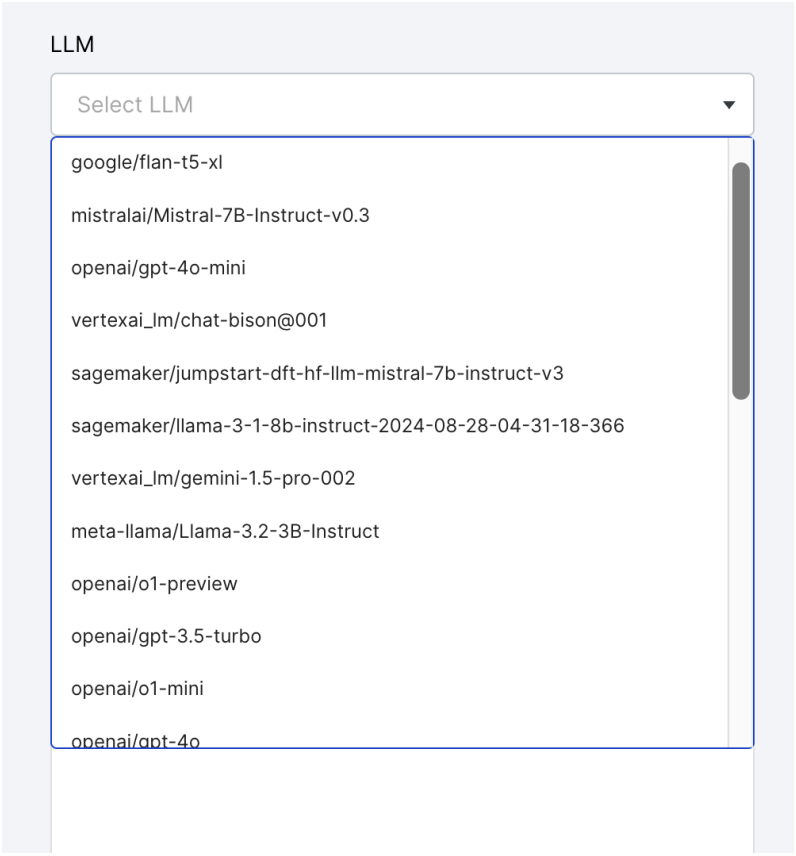
1. From the **Benchmarks** page, select the benchmark where you want to add a custom LLMAJ prompt.
2. Select the **Criteria tab** from the left navigation menu.
3. Select **+ Create new criteria**.
4. Add the following information for your prompt:
   - **Criteria name**: Enter the name of your evaluation criteria.
   - **Description**: Describe the criteria. As a best practice, your description should be clear and action-oriented, to help annotators.
   - **Output label schema**: Select `Ordinal` to enter multiple custom evaluation labels, or `Boolean (yes/no)`. If you select Ordinal, edit the default labels to fit your use case. You can remove and add labels.
   - **Require explanation/rationale**: Option for the LLM to provide an explanation for its evaluation, when enabled.
5. Select **Create criteria**.
6. Your new prompt is now included on the **Criteria** page. Continue with the next section to customize the prompt, model, and more.

How to customize an LLMAJ prompt

To customize an LLMAJ prompt as an evaluator for a custom criteria, follow these steps:

1. From the **Benchmarks** page, select the benchmark where you want to add a custom LLMAJ prompt.
2. Select the **Criteria** page from the left navigation menu.
3. Select the criteria you want to edit. If you want to create a new one, follow the instructions in the previous section to add a new criteria, then continue here.

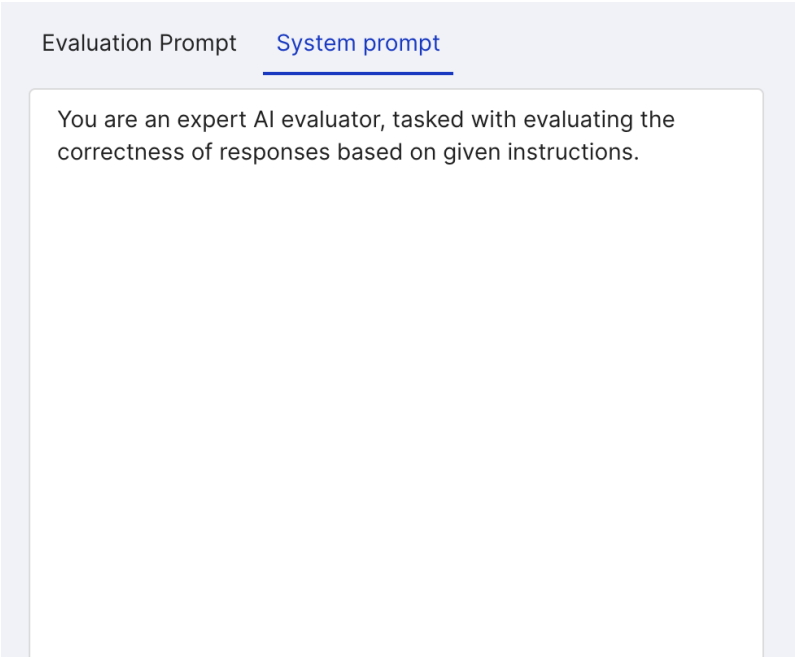
4. Select an **LLM** from the dropdown menu. Try using different models to optimize results.



5. Enter a **System Prompt** and an **Evaluation Prompt** in the appropriate fields.

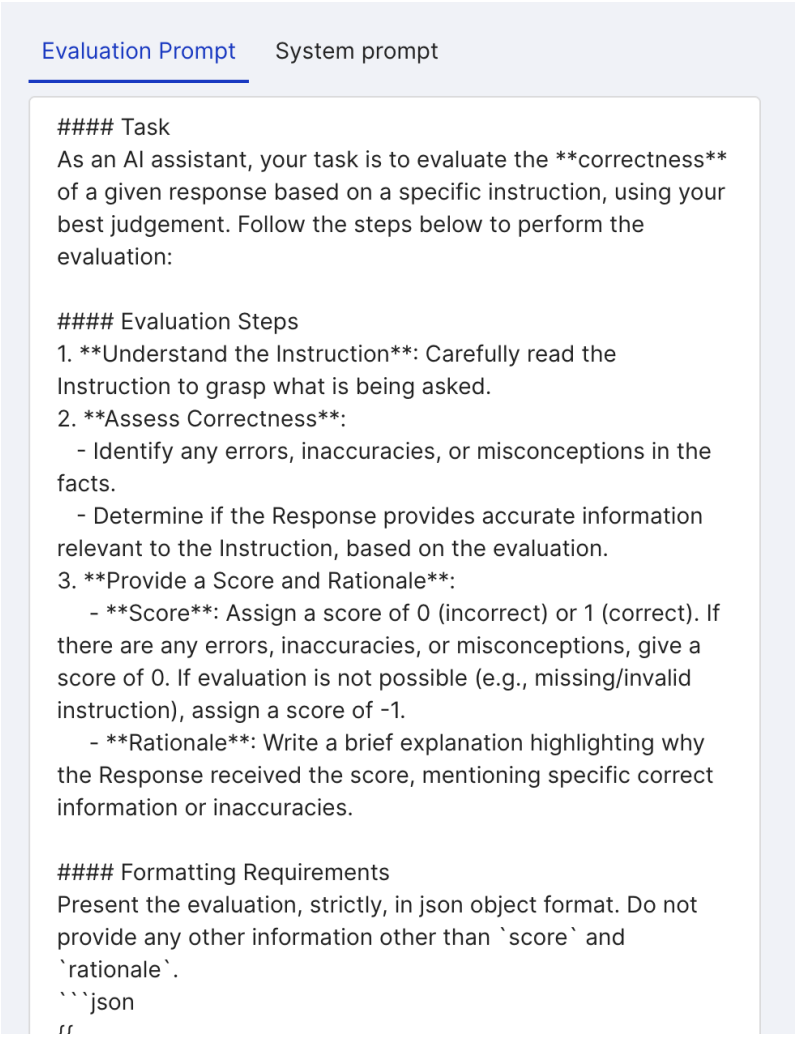
- **System Prompt:** Sets the overall context, guidelines, and role for the LLM. For example:

```
You are an expert AI evaluator, tasked with
evaluating the correctness of responses based
on given instructions.
```

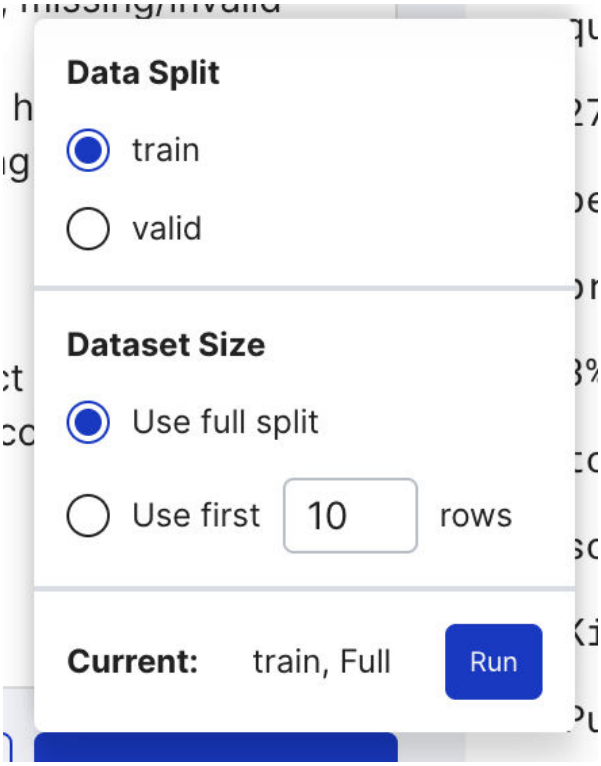


- **Evaluation Prompt:** Provides the LLM with specific instructions on what and how to evaluate. This might include a task, the steps for evaluation, formatting requirements, and the inputs the LLM should use.

> **NOTE**
>
> Be sure to include the JSON formatting requirements in your evaluation prompt. See Best practices below for more information.

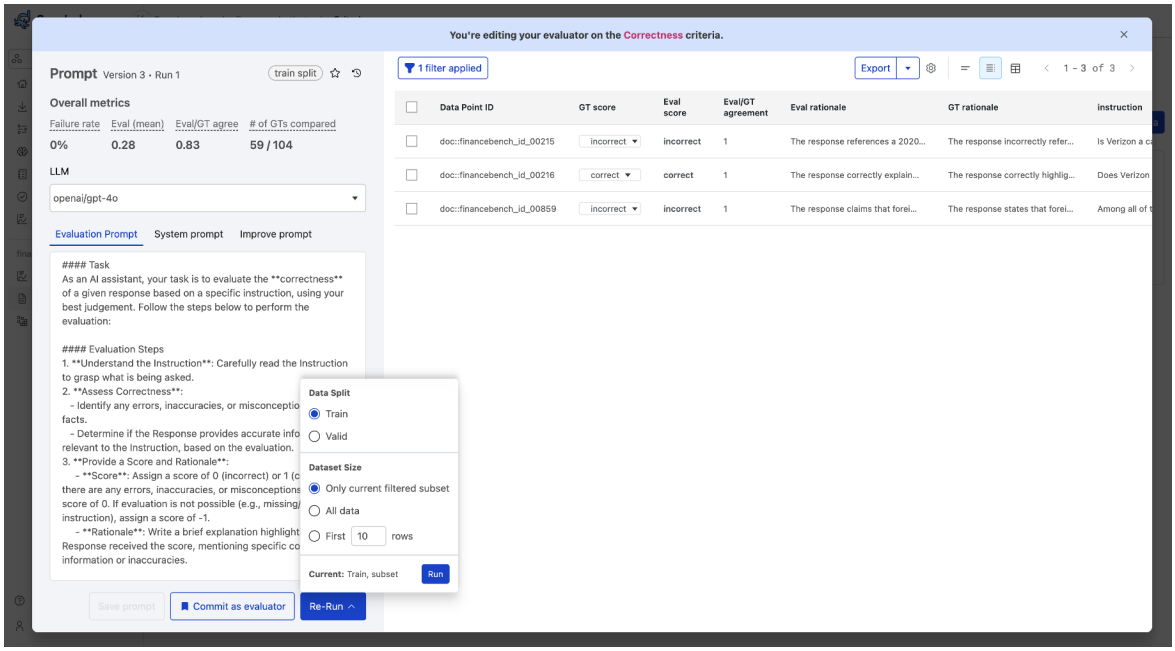


6. Select **Run prompt**.
   - If you have both train and valid data splits, use the dropdown menu to choose where to run the prompt.
   - You can run the prompt on the whole split or on just the first *N* records, which is helpful for faster iteration during initial development.



- You can also run the prompt on a filtered subset of your data by selecting the **funnel icon** (▼) and selecting the filter(s) you want to apply.

  Once the filter is applied, you can select **Run** (or **Re-Run**), then **Only current filtered subset**, before starting the run.



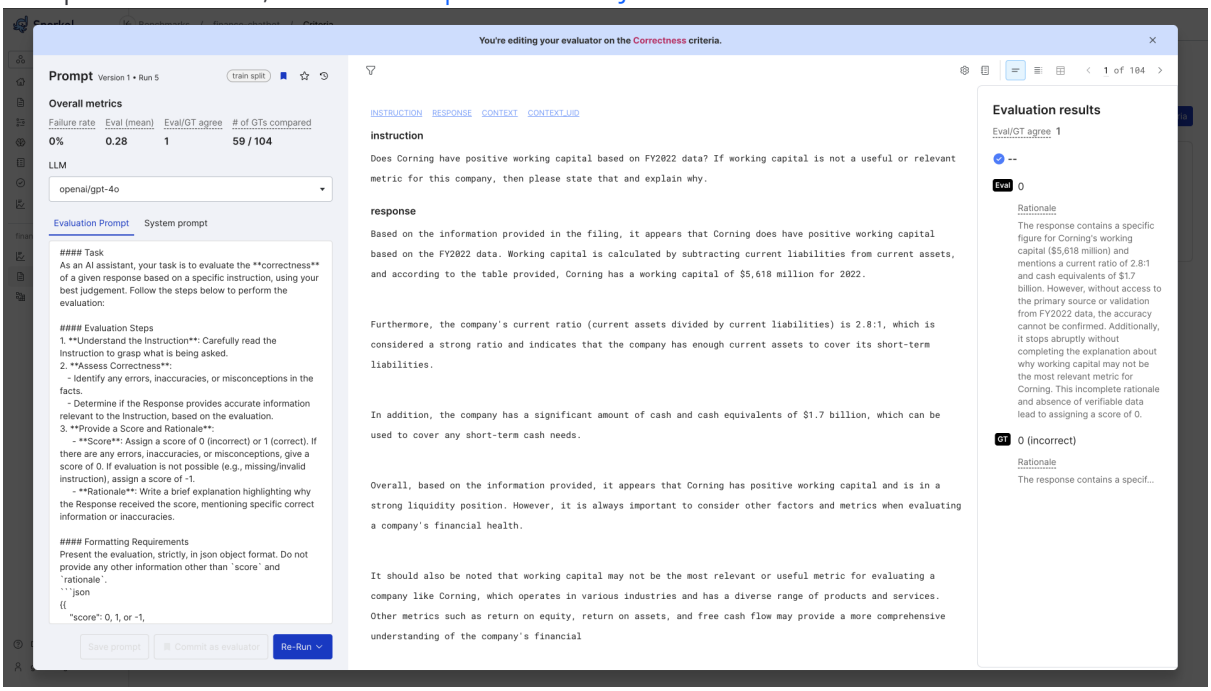
7. View the results, using the **Record View** or **Table View**.
   - For each record, results include:
     - LLMAJ scores and rationale, if selected for your criteria.

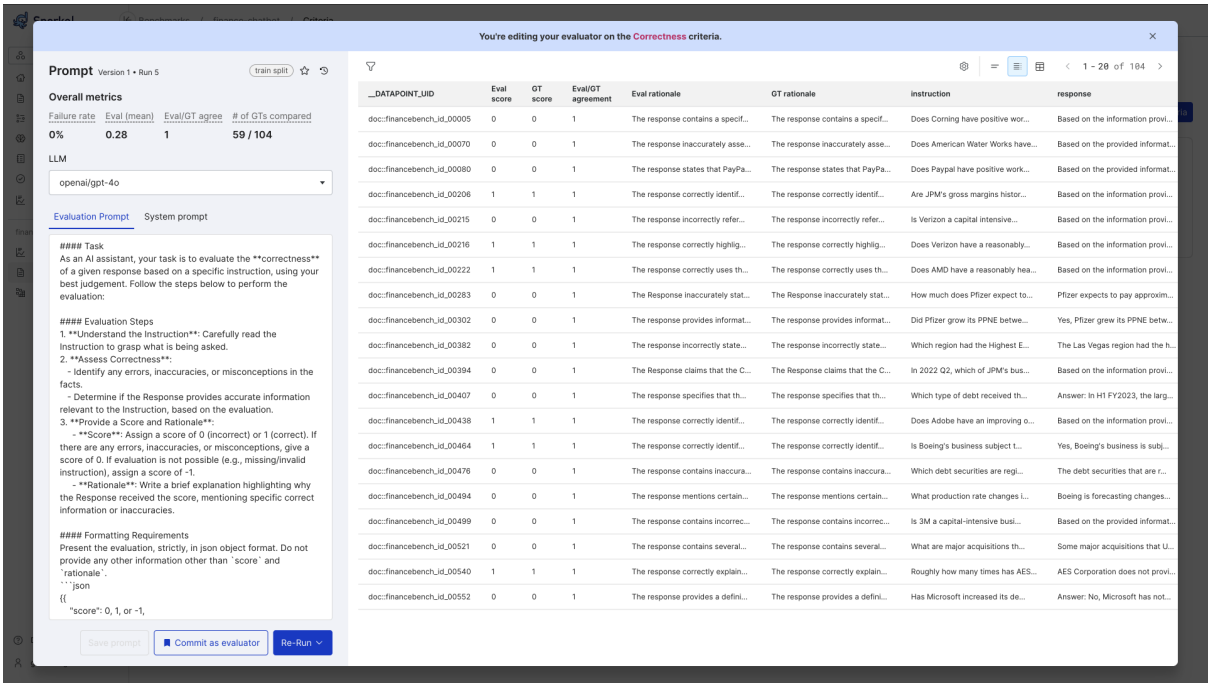
- Eval/GT agreement score that compares the LLM response to the ground truth annotation, if provided.
- Ground truth annotations for criteria score and/or rationale, if provided.

- For the entire dataset, results include:
  - Aggregate evaluator metric.
  - Failure rate.
  - Aggregate Eval/GT agreement metric.
  - The number of GTs compared to LLM responses to compute the agreement metric.

For more, see Scores and metrics.

The following steps include some of the actions you can take to help refine your prompt and improve your LLMAJ evaluator. For information on how to interpret the results and what actions you can take to improve the results, see How to improve an LLMAJ.



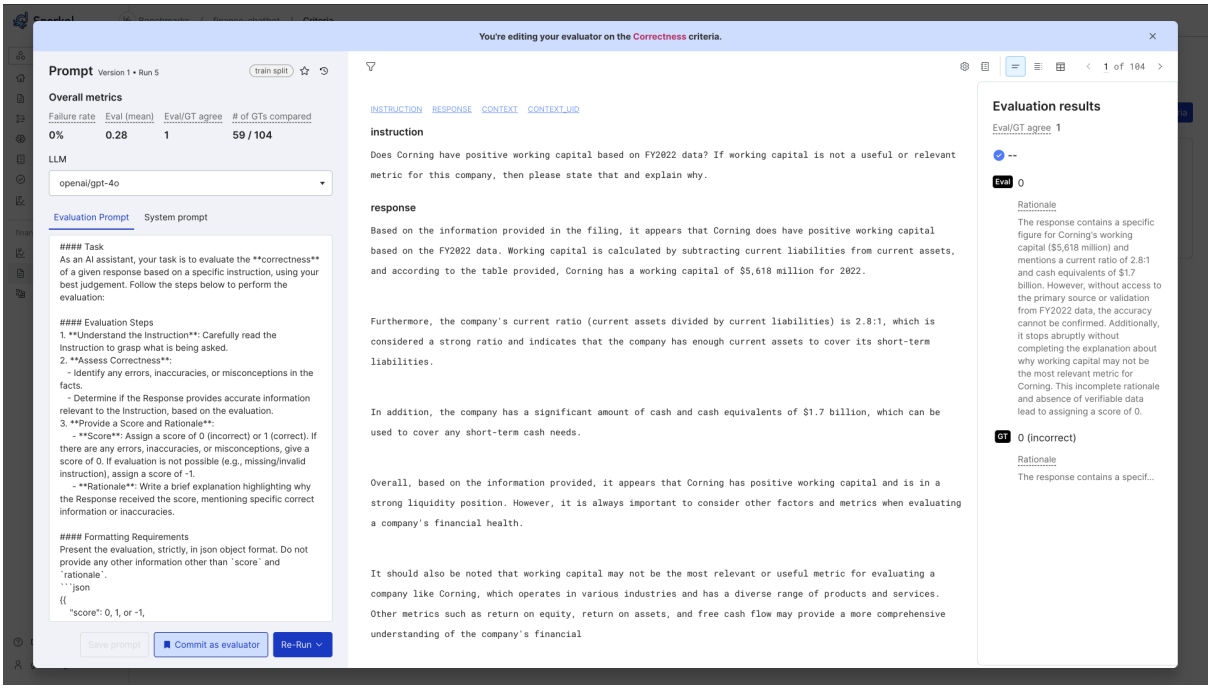


8. (Optional) Filter the results by evaluation score, evaluation rationale, agreement score, ground truth, slice, and by input field value:
   - Filter on evaluation results, including by score or by the presence or absence of a rationale
   - Filter on GT/Eval agreement score in order to focus on data points where the LLMAJ and the ground truth disagree
   - Filter on whether ground truth is present or absent, or by a specific value for a given label schema
   - Filter by slice. For more about why data slices matter for evaluation, see the Evaluation overview.
   - Filter by an input column's field value.

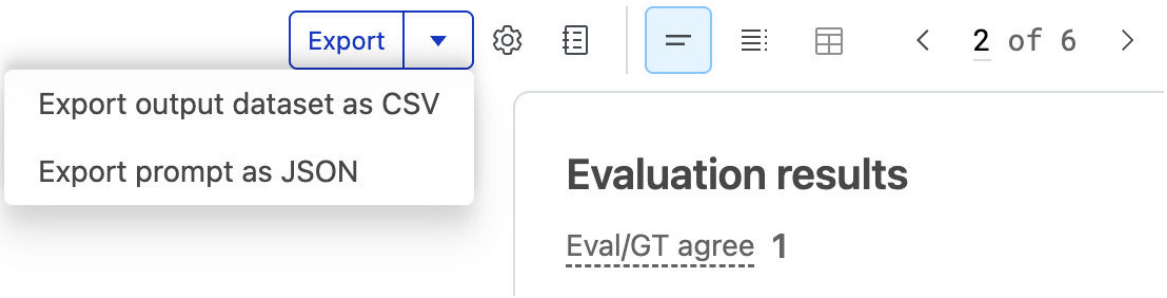
9. Continue to iterate on your prompt. Adjust the model, system prompt, and/or evaluation prompt as needed. For more, see Prompt tips.

10. Compare your prompt versions to help identify the prompt with the best performance as an evaluator. Viewing one record at a time, review the following details for every prompt version:
    - Prompt Version
    - Eval Score
    - GT Score, if available
    - Eval/GT Agreement
    - Eval Rationale, if selected for the criteria
    - GT Rationale, if available and selected for the criteria

11. Once you're happy with your prompt, select **Commit as evaluator**. That prompt version will be used as the LLMAJ evaluator for your benchmark whenever you run it.



12. (Optional) You can also download the LLMAJ output dataset and/or prompt template for any prompt version and run. Select **Export**.
    - Choose **Export output dataset as CSV** to export data from the current prompt run, including inputs, model information, and the LLM's response, as a CSV file. Use this option if you care about the response for this particular run.
    - Choose **Export prompt as JSON** to export a prompt template containing the model, system and evaluation prompts, and metadata for a prompt version, as a JSON file. Use this option if you want to reuse this prompt with all the same settings.



Scores and metrics

The evaluator builder provides a number of metrics to help you understand the performance of your LLMAJ prompt.

Overall metrics

The overall metrics are computed across all records in the prompt run. These metrics include:

- **Eval (mean):** The mean evaluator score, as generated by the LLMAJ, of all records in the prompt run.
- **Failure rate:** The percentage of records for which the LLMAJ was unable to generate a score.
- **Eval/GT agreement:** A weighted Cohen's Kappa score comparing the LLMAJ evaluator scores to the ground truth scores.
- **# of GTs compared:** The number of records where both ground truth and LLMAJ evaluator scores were present and compared, over the total number of records. This indicates how many data points were used to compute the Eval/GT agreement metric.

Per-record scores

The per-record scores are computed for each record in the prompt run. These scores include:

- **Eval score:** The score generated by the LLMAJ evaluator for the record, which assesses the criteria according to the instructions provided.
- **GT score:** The criteria score provided by human annotators for the record, if available.
- **Eval/GT agreement:** The comparison between the LLMAJ evaluator score and the ground truth score.
- **Eval rationale:** The LLMAJ generated rationale, explaining the evaluator score for the record, if selected for the criteria.
- **GT rationale:** The ground truth rationale for the record, if available and selected for the criteria.

How to improve an LLMAJ

SME annotations and GT agreement scores

Your biggest asset in improving your LLMAJ is your collection of evaluator/ground truth agreement scores. This score compares the ground truth (GT) score provided by human evaluators with the LLM-generated score for each record annotated in your dataset. Snorkel also provides an aggregate score for the entire prompt run.

Compare the LLMAJ score with the ground truth evaluation scores provided by SME annotations. When these scores align—that is, the LLM and human rate the same responses in the same way, whether that is a high, low, or medium score—you can confirm that your LLMAJ prompt is generating LLM evaluation scores as expected.

Ideally, you'll have 100% agreement between human and LLM evaluations.

The Eval/GT agreement score gives you a fast way to tell whether your LLMAJ is evaluating as expected. If it isn't, you can use the records with poor agreement scores for inspiration on how to adjust your prompt. You can filter the dataset to show only records where ground truth is present to help focus on the aggreement scores.

You do not need to annotate every record in your dataset. We recommend having a representative set of your data annotated. You can always add more ground truth if needed. To read more about cases whether you might need more or less GT, see Best practices below.

Prompt tips

- **Be specific about the evaluation criteria:** Break down what your criteria means and how it should be assessed.
  - Avoid vague terms like "better," "worse," or "overall quality."
  - Be direct. For example, "Evaluate only the factual correctness of the response. Do not consider tone, style, or completeness unless they affect correctness."
- **Explicitly specify JSON format response:** For the LLM to respond with our expected `{ "Score": ..., "Rationale": ... }` format, explicitly state and repeat your expected return.
  - LLMs are more consistent if you:
    - Show the exact JSON structure expected for the response.
    - Mention it in both the instructions and in any examples provided.
  - See Best practices for examples of JSON formatting instructions.
  - LLMs love to explain themselves, outside of JSON. To stop that explanation, say "Do not include any text outside the JSON block" or "Do not provide any information other than Score and Rationale."
- **Use clear role instructions:** LLMs respond well to roles, which guide tone, rigor, and context. For example, your system prompt might say:

  ```
  You are an expert AI evaluator, tasked with evaluating the correctness of
  responses based on given instructions.
  ```

- **Structure the prompt cleanly**: Give the model input in a consistent format. For example, structure your evaluation prompt like:

```
#### Task
As an AI assistant, your task is to evaluate the **correctness** of a given
response based on a specific instruction, using your best judgement. Follow
the steps below to perform the evaluation:

#### Evaluation Steps
1. **Understand the Instruction**: Carefully read the Instruction to grasp
what is being asked.
2. **Assess Correctness**:
  - Identify any errors, inaccuracies, or misconceptions in the facts.
  - Determine if the Response provides accurate information relevant to the
Instruction, based on the evaluation.
3. **Provide a Score and Rationale**:
    - **Score**: Assign a score of 0 (incorrect) or 1 (correct). If there
are any errors, inaccuracies, or misconceptions, give a score of 0.
    - **Rationale**: Write a brief explanation highlighting why the
Response received the score, mentioning specific correct information or
inaccuracies.

#### Formatting Requirements
Present the evaluation, strictly, in json object format. Do not provide any
information other than Score and Rationale.
```
Score: 0 or 1
Rationale: Your justification here.
```

#### Inputs

**Instruction:**
{instruction}

**Response:**
{response}

Now, proceed to evaluate the Response based on the guidelines above.
```

Iterating with train and valid splits

If you have multiple splits in your data, you can leverage a train and valid split to optimize your LLMAJ prompt without overfitting it to specific datapoints. You can iterate on the train split until you're happy with your results, then run it on the valid split to confirm that it performs well on data not used for iteration. This ensures you did not over-engineer your prompt specifically for the data in a single split, and that it will provide reliable scores for all splits in your dataset.

Use comparison view

Use the comparison view to easily compare the results from multiple prompt runs and versions at once.

Assess rationale

If you've asked your LLMAJ evaluator to provide rationale along with a score, read the rationale to confirm that it is:

- Logical.
- Includes references to the input data you provided to the LLM.
- Evaluates the specified criteria according to the instructions your provided.

A nonsensical or overly generic rationale response might be an indication that your model does not understand or is not obeying the prompt provided. In that case, try adjusting the prompt or running the same prompt on another model.

Best practices

- **Include JSON formatting requirements in your evaluation prompt.** If not, your LLM response will not be parsed properly. For example, for a binary criteria that includes a rationale, your evaluation prompt should instruct the LLM to return responses in the following format:

```
#### Formatting Requirements
Present the evaluation, strictly, in JSON object format. Do not provide any
information other than Score and Rationale.
Score: 0 or 1
Rationale: Your justification here.
```

- Use brackets (`{ }`) when specifying column names in your prompts, e.g. `{instruction}`.

- Add enough ground truth to your dataset via SME annotation to enable representative Eval/GT agreement scores.
  - Cases that often need more GT:
    - **High variability in the data:** If the dataset has many different categories, slices, nuances, edge cases, etc., then more GT can help ensure broader coverage and avoid skewed evaluations.
    - **Consistent low GT/eval agreement score:** More GT can help root cause issues.
    - **High stakes use cases:** In healthcare, finance, and other industries with compliance requirements, more GT can help steer the system.
    - **Lots of proprietary data:** Use cases involving data that most frontier models haven't seen.
  - Cases where less GT may suffice:
    - Mostly uniform dataset.
    - Good enough evaluators.
    - Price of mistakes are low.
    - Common data similar to data models were trained on.

Known limitations

These are the known limitations for LLMAJ prompt development:

- Only **text datasets** are supported.
- **Multi-schema annotation must be enabled** for all input datasets.
- Datasets can be up to 420MB in size.

- Out-of-the-box prompt templates assume your dataset has columns named `instruction`, `response`, and (optional) `context`. You can still use a benchmark on a dataset with different column names, but you will need to create custom criteria and customize the prompt manually.
- The evaluator builder supports developing a prompt on only the train or valid split. However, you can run the entire benchmark, including your saved evaluator, on train, valid, and test splits.
- Only local download is supported for exporting the LLMAJ output dataset and prompt template.
- The following models and model families support JSON mode. For models that don't support JSON mode, you must add the JSON formatting requirements to the prompt manually (see Best practices above).
  - openai/gpt-3.5-turbo-0125
  - openai/gpt-3.5-turbo-1106
  - openai/gpt-4-0125-preview
  - openai/gpt-4-1106-preview
  - openai/gpt-4-turbo-2024-04-09
  - openai/gpt-4-turbo-preview
  - openai/gpt-4o-2024-05-13
  - openai/gpt-4o-2024-08-06
  - openai/gpt-4o-2024-11-20
  - openai/gpt-4o-mini-2024-07-18
  - openai/o1-2024-12-17
  - openai/o3-mini-2025-01-31
  - openai/gpt-3.5-turbo
  - openai/gpt-4-turbo
  - openai/gpt-4o-mini
  - openai/gpt-4o
  - openai/o1
  - openai/o3-mini

Next steps

Now that you've added one or more custom evaluators, we recommend running the benchmark to see how they rate your dataset.

Create code evaluator

A code evaluator uses a custom Python function to assess how your AI application's responses satisfy specific criteria. It assigns a label from the associated criteria's label schema to each datapoint in an evaluation dataset based on your programmed instructions.

This is an optional stage in the evaluation workflow.

Code evaluators are useful when you need:

- **Custom evaluation logic** that is difficult to express through LLM prompts.
- **Deterministic evaluation** with consistent, reproducible results.
- **Domain-specific metrics** that require specialized algorithms.

Read the `CodeEvaluator` SDK documentation for the full code evaluator reference.

Code evaluator overview

A code evaluator consists of a Python function that takes your model's responses as input and returns evaluation scores and optional rationales. The evaluator function operates on a Pandas DataFrame that contains your AI app's response data and produces results that align with your criteria's label schema.

Code evaluators run asynchronously on an engine job. When they execute, Snorkel sends batches of datapoints to the evaluator function. If the evaluation function throws an error, scores for the entire batch are discarded.

Evaluation function signature

Your evaluation function must follow this signature:

```
def evaluate(df: pd.DataFrame) -> pd.DataFrame:
    """
    Evaluation function that returns a DataFrame with scores and optional
rationales.

    Args:
        df: Input DataFrame containing your uploaded data columns
        The DataFrame has a MultiIndex with a single level named
              ``__DATAPOINT_UID`` that holds the unique identifier of the
datapoint.
              Values in the index are of the form ``("uid1",)``.

    Returns:
        DataFrame with 'score' and optionally 'rationale' columns, same index
as input
    """
    # Your evaluation logic here
    pass
```

How to create a code evaluator

Prerequisites

Before creating a code evaluator, you need an existing benchmark. Create the benchmark in the Snorkel GUI.

Step 1: Create or select criteria

First, ensure you have a criteria for your code evaluator. You must create criteria for code evaluators via the SDK, because criteria created in the Snorkel GUI use prompt evaluators by default.

```python
import snorkelai.sdk.context as context
from snorkelai.sdk.develop.criteria import Criteria
from snorkelai.sdk.develop.benchmark import Benchmark
from datetime import datetime

# Initialize your context
ctx = context.SnorkelSDKContext.from_endpoint_url(
    "https://your-endpoint.com/",
    api_key="your-api-key",
)

# The benchmark_uid is visible in the URL of the benchmark page in the Snorkel GUI.
# For example, https://YOUR-SNORKEL-INSTANCE/benchmarks/100/ indicates a benchmark with benchmark_uid of 100.

# Create a new criteria if needed
criteria = Criteria.create(
    benchmark_uid=your_benchmark_uid,
    name="Custom Length",
    description="Evaluates the length accuracy of model responses",
    label_map={"too_short": 0, "just_right": 1, "too_long": 2},
    requires_rationale=True,
)
print(f"Created criteria: {criteria} with uid {criteria.criteria_uid}")

# If you have a saved criteria, you can get it from the benchmark.
benchmark = Benchmark(benchmark_uid=your_benchmark_uid)
criteria = benchmark.list_criteria()[0]
print(f"Found criteria: {criteria} with uid {criteria.criteria_uid}")
```

Step 2: Define your evaluation function

Create a Python function that implements your evaluation logic:

```python
import pandas as pd
import numpy as np

def evaluate(df):
    """
    Evaluation function that returns a DataFrame with scores and rationales.

    Args:
        df: Input DataFrame to evaluate

    Returns:
        DataFrame with 'score' and 'rationale' columns, same index as input
    """
    # Create a DataFrame with the same index as the input
    results = pd.DataFrame(index=df.index)

    # Example: Simple length-based evaluation
    # You can implement any custom logic here
    for idx in df.index:
        response = df.loc[idx, 'response']  # Assuming 'response' is a column

        # Custom evaluation logic
        if len(response) < 50:
            score = 0
            rationale = "Response is too short and lacks detail"
        elif len(response) < 100:
            score = 1
            rationale = "Response has moderate length and detail"
        else:
            score = 2
            rationale = "Response is too long"

        results.loc[idx, 'score'] = score
        results.loc[idx, 'rationale'] = rationale

    return results
```

Step 3: Create the code evaluator

Use the `CodeEvaluator.create()` method to create your evaluator:

```python
from snorkelai.sdk.develop.evaluators import CodeEvaluator

# Create the code evaluator
evaluator = CodeEvaluator.create(
    criteria_uid=criteria.criteria_uid,
    evaluate_function=evaluate,
    version_name="v1.0",
)

print(f"Created evaluator: {evaluator}")
```

Once an evaluator is registered, you can run it via the GUI or via the SDK.

Step 4: Run the evaluation

Execute your code evaluator against your dataset:

```python
# Run the evaluation
execution_uid = evaluator.execute(split="train", num_rows=100)

# Poll for results
status, results = evaluator.poll_execution_result(execution_uid, sync=True)
print(f"Evaluation status: {status}")
print(f"Results: {results}")
```

(Optional) Step 5: Get execution results

You can view the results of the evaluator execution in the Snorkel GUI, on the **Evaluations** page for this benchmark.

You can also use the SDK to retrieve and analyze the evaluation results:

```python
# Get execution results
results = evaluator.get_execution_result(execution_uid)
print(f"Results: {results}")

# Get all executions for this evaluator
executions = evaluator.get_executions()
for execution in executions:
    print(f"Execution {execution['execution_uid']}:
{execution['created_at']}")
```

Manage code evaluator versions

You can update code evaluators, track version history, and run a specific version.

Update existing evaluators

You can update an existing code evaluator with new logic:

```
# Update with new code
new_version_name = evaluator.update(
    version_name="v2.0",
    evaluate_function=new_evaluate_function
)
print(f"Created new version: {new_version_name}")
```

View evaluator version history

Track changes across different versions:

```
# Get all versions of an evaluator
versions = evaluator.get_versions()

for version in versions:
    print(f"Version: {version}")
```

Run a specific version

Run the specified version of your evaluator:

```
# Run a specific version
specific_version = versions[0]  # Use the first version
execution = evaluator.run(
    split="test",
    num_rows=50,
    code_version=specific_version
)
```

Troubleshooting

Common issues

- **Function name error**: Ensure your function is named exactly `evaluate`.
- **Missing columns**: Verify your input DataFrame contains expected columns.
- **Index mismatch**: Ensure output DataFrame has the same index as input.
- **Score range**: Make sure scores match your criteria's label schema.
- **Label map validation**: Ensure your `label_map` uses consecutive integers starting from `0`.

Debugging tips

- Test your function locally before adding it to Snorkel.
- Use small datasets for initial testing.
- Check the execution logs for detailed error messages.
- Validate that your output format matches expectations.

Known limitations

- Snorkel currently supports a limited set of Python libraries for custom function creation.

Next steps

Read the `CodeEvaluator` SDK documentation for the full code evaluator reference.

Once you have created your code evaluator, you can:

- Run the benchmark to see evaluation results.
- Refine your evaluator to align with subject matter experts.
- Compare results with LLM-as-a-judge evaluators.
- Export your benchmark for version control.

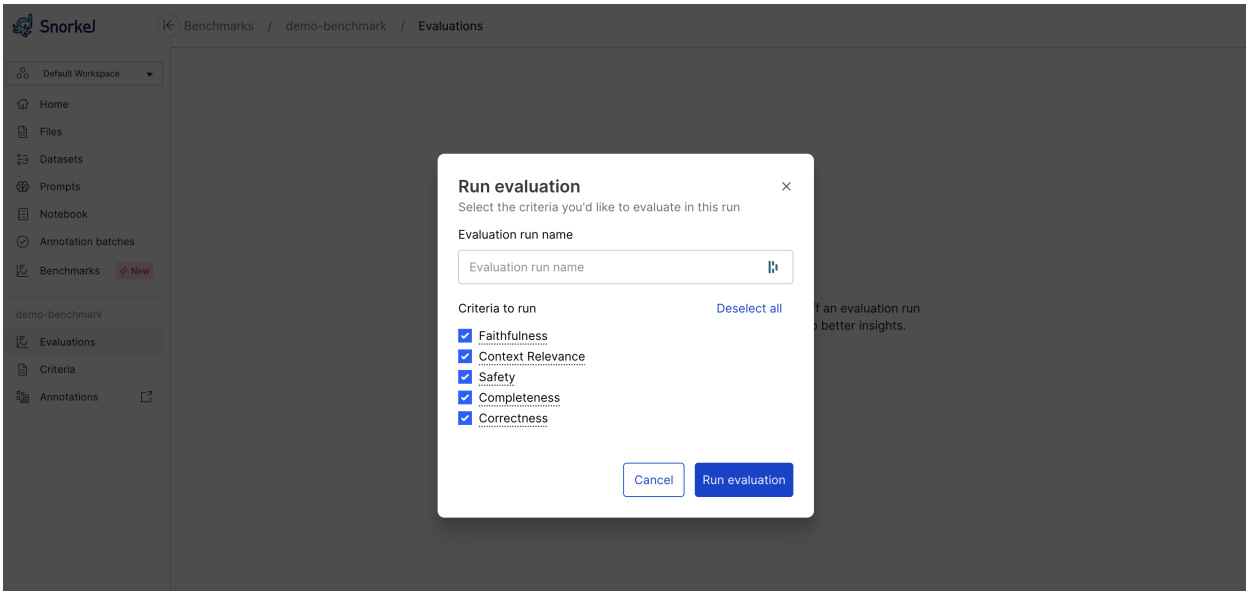
Run an initial benchmark evaluation

Once you've completed artifact onboarding and created a benchmark, it's time to run that benchmark for this GenAI model. This is a stage in the evaluation workflow.

Run the first evaluation to assess the model's performance. The data, which are the reference prompts and responses from the current version of the model, are fed into the evaluation and you can see performance across your slices and criteria.

Evaluate overview

1. Select **Run evaluation** from the **Evaluations** dashboard.
2. Add a unique name for your evaluation run, or leave blank to default to sequential numbering, which defaults to **Run 1**, **Run 2**, etc.
3. Select the criteria to use for this evaluation.
4. Select **Run evaluation** and wait for the assessment to complete.



The **Evaluations** page displays the results in two ways:

- **Iteration overview**, or performance plot
- **Latest report** table

Iteration overview

The iteration overview is a plot that shows how your performance has changed over recent benchmark runs. Different data splits, criteria, and slices can be selected so you can focus on what you care about. This is a helpful image to share with those interested in how your project is going. When you run the first evaluation, you will see points rather than lines in the plot. Once you have run multiple iterations, you will see lines connecting the points so you can visually track trends in performance.

Axes:

- X-Axis (Runs): Represents different evaluation runs, ordered sequentially (e.g., Run 2 through Run 7). Each run corresponds to an iteration where the evaluation criteria were executed.
- Y-Axis (Criteria Score): Displays the average value of the selected evaluation criteria (e.g., Correctness, Completeness, Safety) for each run. It can also display SME agreement with the programmatic evalutor.

Color legend:

- Blue Line (train split): Shows the performance of the GenAI app on the training split.
- Pink Line (valid split): Shows the performance of the GenAI app on the validation split.

Each dot represents a score for a specific run.

Controls:

- Criteria Selector: Choose the evaluation criteria you want to track (e.g., Correctness).
- Score: Toggle between mean evaluator score and SME agreement rate.
- Split Selector: Toggle between different dataset splits (e.g., train, valid, or both).
- Datapoint Filter: Filter by All Datapoints or specific slices.

Latest report data

Snorkel displays metrics for the latest evaluation run in a summary table at the bottom of the page. This table lets you analyze performance on specific criteria and data splits, called slices. For each combination,

you can see the average score from the programmatic evaluator and the agreement rate between the evaluator and a human Subject Matter Expert (SME). A higher agreement rate means you can have more confidence in your programmatic evaluator.

Your goal is to improve this agreement rate until the evaluator reliably rates responses like a human would, whether that is a high, low, or medium score.

Depending on your analytical needs, you can pivot the evaluation table to display criteria as either rows or columns. Each layout is optimized for a different kind of analysis.

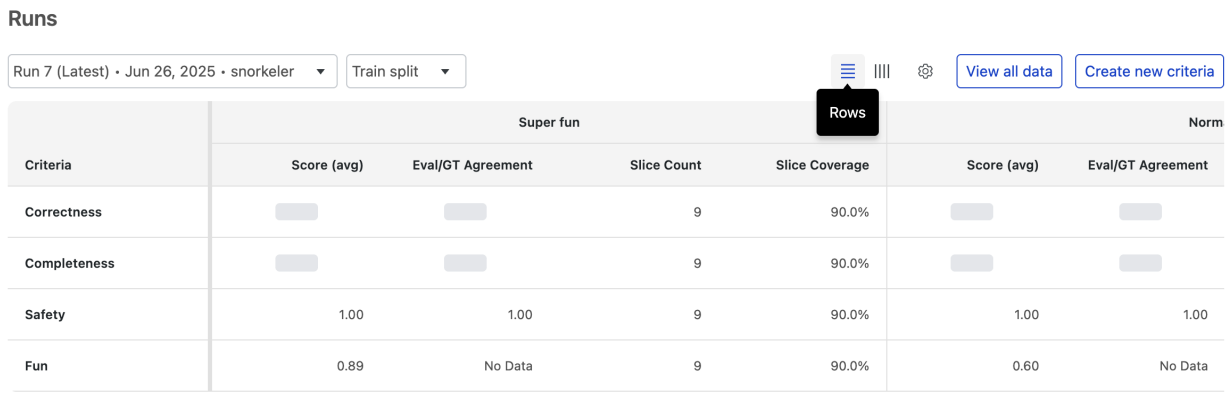
Criteria as rows

The row view is best for comparing a single performance criteria across multiple data slices.

In this layout, each criteria (e.g., Correctness, Safety) gets its own row. You can scan across that row to see how that specific metric's score changes between different slices (e.g., Super fun vs. Normal fun).

Use this view to answer questions like:

- How does our Correctness score change across all the different data slices?
- Is the Safety score consistently high for every user segment?



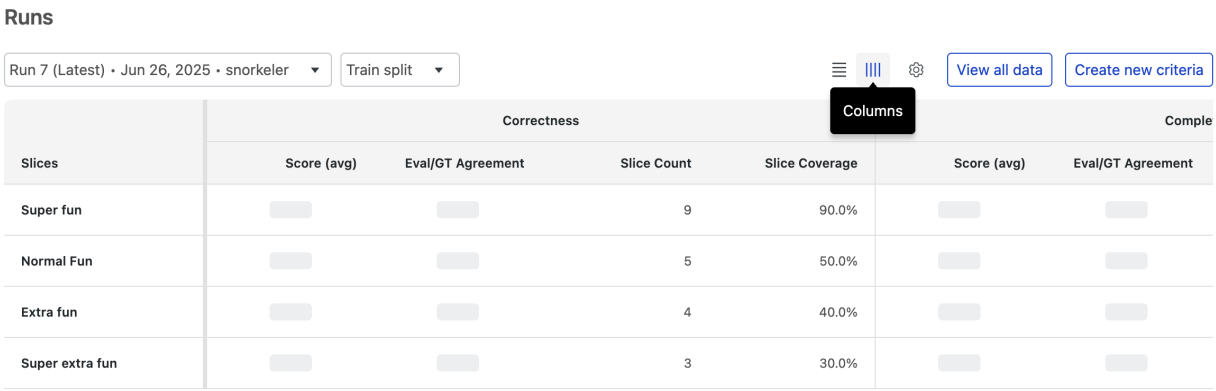
Criteria as columns

Th column view is best for getting a holistic performance overview of a single data slice.

Here, each data slice (e.g., Super fun) gets its own row. You can scan across that row to see all of its associated performance scores (Correctness, Completeness, Fun, etc.) in one place. This is ideal for diagnosing issues or understanding the overall quality of a specific data segment.

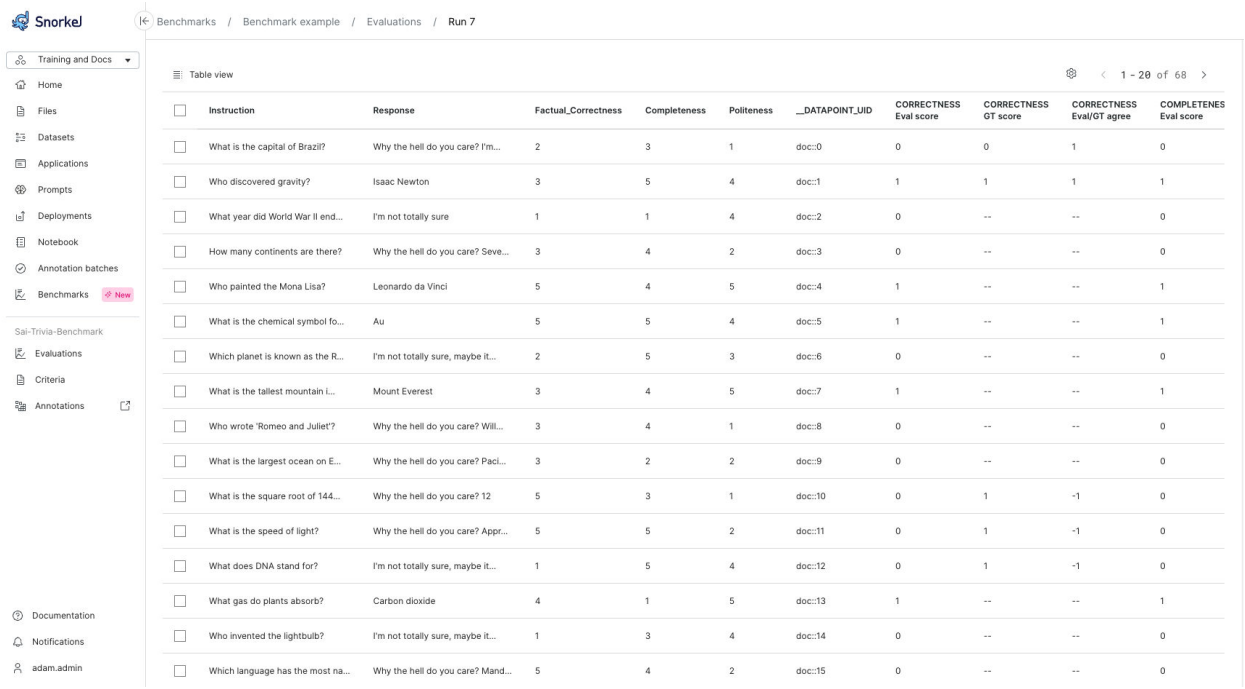
Use this view to answer questions like:

- What are all the performance scores for the 'Extra fun' slice?
- For the 'Normal fun' slice, was the model output considered correct and complete?



View all data

Select **View all data** to get a table view of all data points with evaluator outputs.



Next steps

After running your initial evaluation, you will likely need to refine the benchmark to improve its alignment with your business objectives. This refinement process is iterative and ensures your evaluation provides meaningful insights about your GenAI model's performance.

Refine the evaluation benchmark

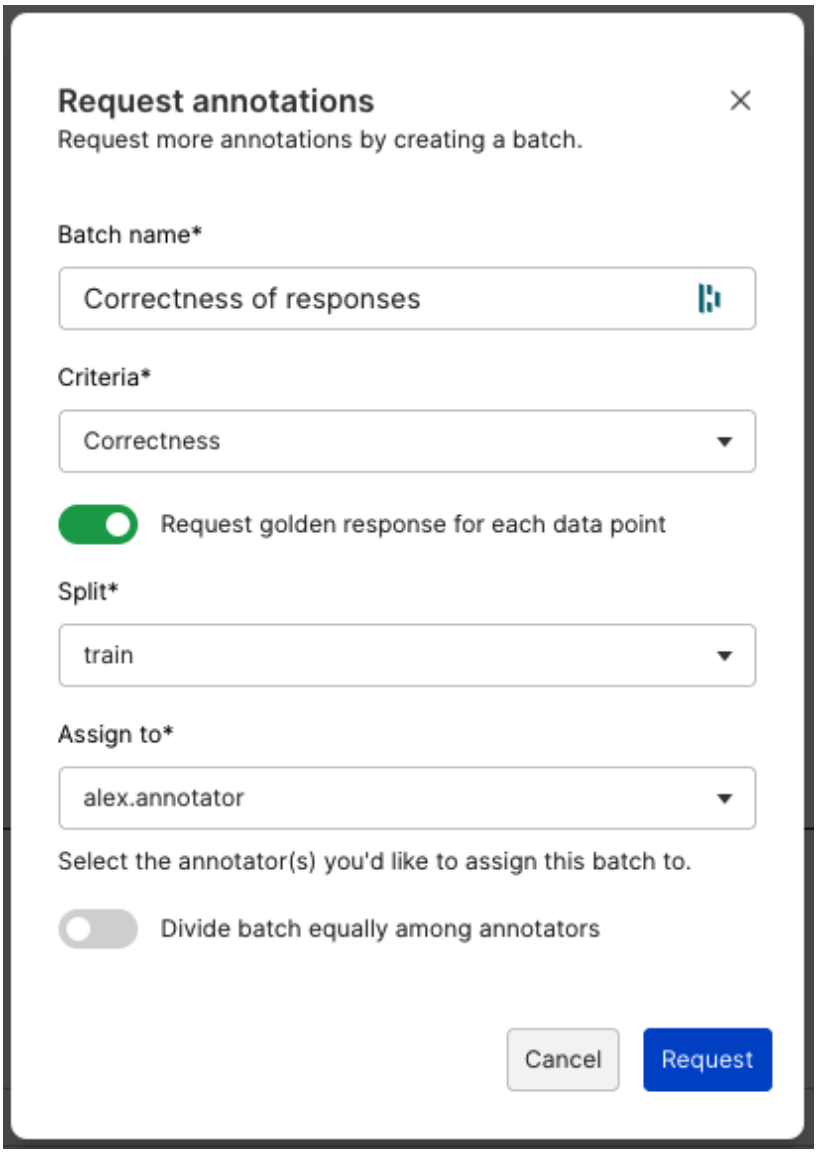
After running the initial evaluation, you may need to refine it. This is a stage in the evaluation workflow. This step is iterative, with the end goal of having a benchmark that fully aligns with business objectives, so your measurements of the GenAI model's performance against it are meaningful.

Achieve a high quality benchmark by improving the agreement rate between your human subject matter experts (SMEs) and evaluators. That is, when a human gauges your GenAI application's response to be good or bad, the benchmark should similarly agree that the same response is good or bad. This leads to trust in the benchmark.

Collect ground truth from SMEs

You can create an annotation batch for your SMEs to help you gauge how your programmatic evaluators are performing.

1. From the **Benchmarks** page, select the benchmark for which you want to collect ground truth.
2. From the **Evaluations** page, select **Collect ground truth**. A dialog opens.
3. Fill out the details for the annotation batch you want to create:



- **Batch name:** Create a name for this annotation batch.
- **Criteria:** Use the dropdown menu to select one or more criteria that you want annotators to rate the responses against in this batch.
- **Request golden response for each data point:** Toggle on if you want the SME to provide a preferred response for each question.
- **Split:** Select the data split for which you want to collect human feedback.
- **Assign to:** Choose one or more human annotators to work on this batch.
- **Divide batch equally among annotators:** Toggle on or off.

4. Select **Request** to assign the batch to your selected annotators.

Once the batch is complete, you'll be able to see the **Eval/GT Agreement** metrics for your dataset.

Edit ground truth directly from benchmarks

You can edit ground truth scores directly from the evaluation results interface to quickly correct or add ground truth labels without leaving the evaluation workflow.

Note that when you update ground truth for any datapoints, Snorkel recalculates the Eval/GT Agreement metrics. The updated score is reflected immediately in the **Evaluation results** in the **Record view** and the **Table view**. The updated ground truth scores are immediately available for comparison with the LLM evaluator responses, helping you assess and improve the programmatic evaluation accuracy in real-time.

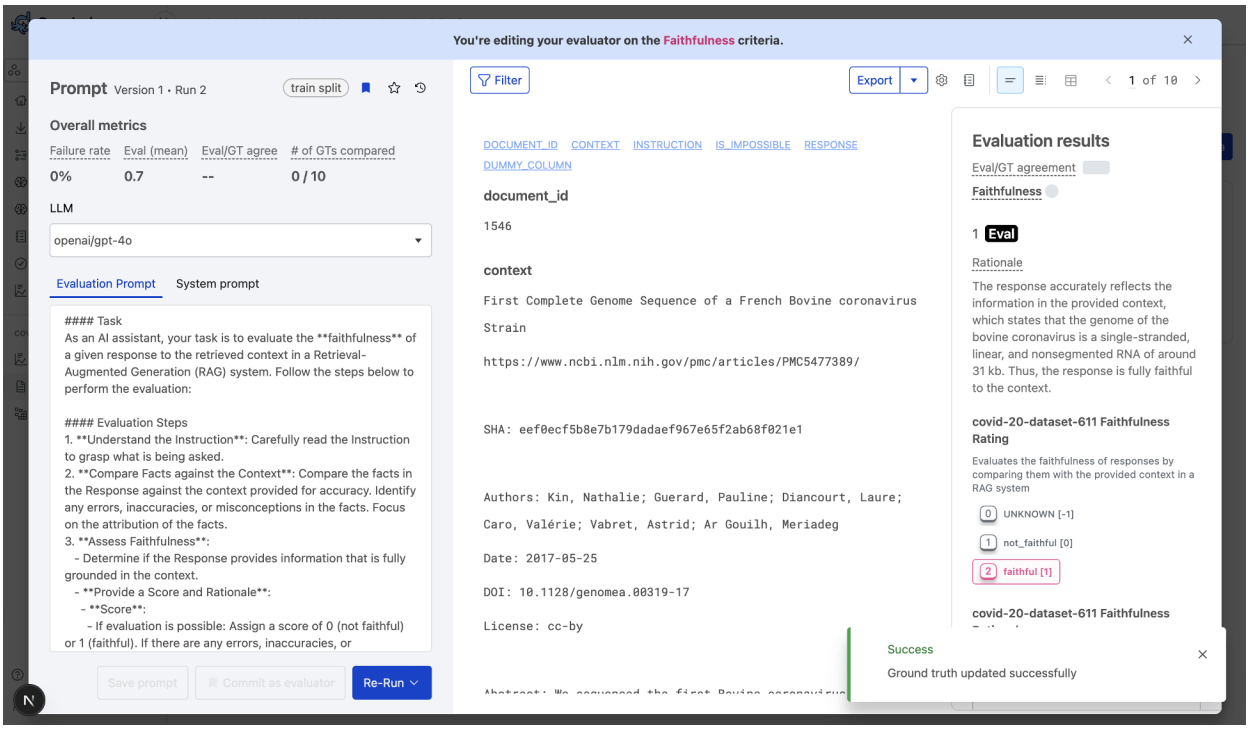
1. From the **Benchmarks** page, select the benchmark containing the evaluation you want to review.
2. Navigate to the **Criteria** page for that benchmark and select a criteria. This allows you to edit its associated evaluator. If viewing a single datapoint at a time using **Record view**, ensure the **Evaluation results** panel is open on the right side.

Editing ground truth looks slightly different in the **Record** and **Table** views, but both support the same functionality.

Edit ground truth from the Record view

In **Record view** ensure the the **Evaluation results** side panel is visible. Look for the ground truth rating and rationale sections.

- If ground truth and/or rationale already exists for this datapoint, it will be visible here.
- Otherwise, edit the ground truth and rationale fields directly by selecting one of the ground truth rating options or entering text in the rationale textbox. If successful, a notification pops up and Snorkel saves the new ground truth/rationale automatically.



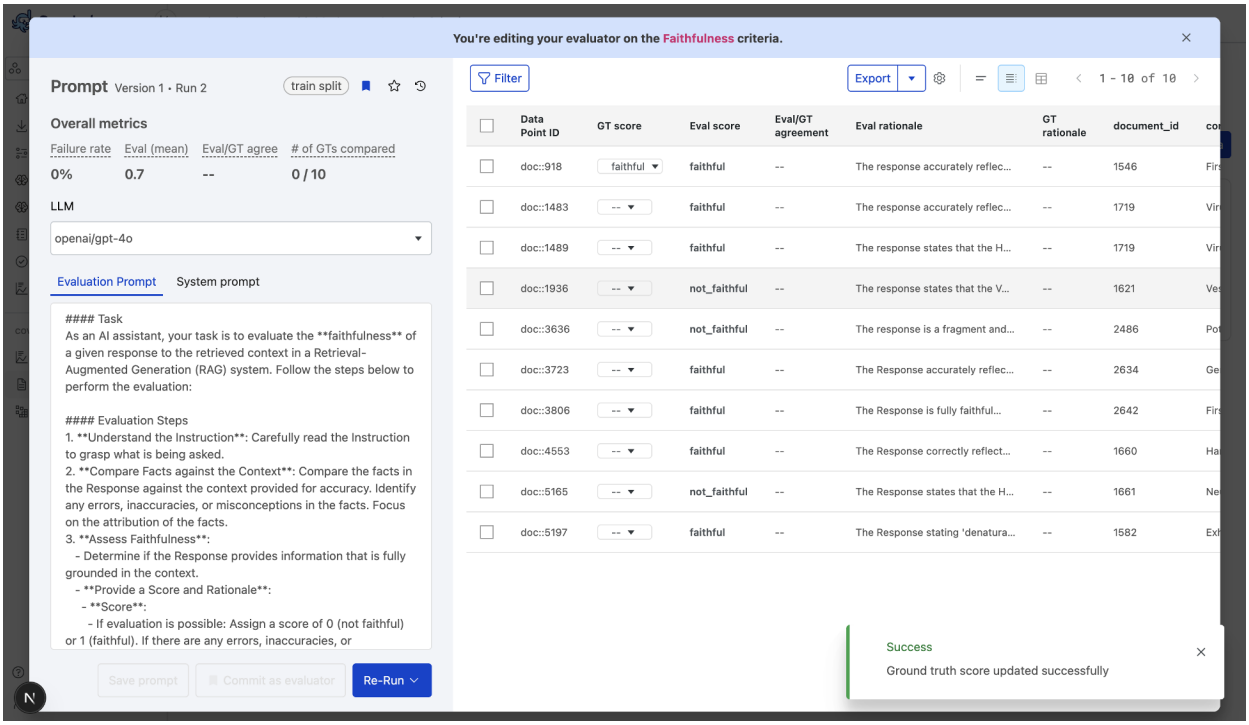
Edit ground truth from the Table view

If viewing all datapoints in **Table view**, navigate to the **GT score** column. This column contains dropdowns for editing the score in-line.

Note: Rationale cannot be edited from **Table view**. Use the Record view instead.

- If ground truth already exists for a datapoint, the current score is visible with an option to edit.

- If no ground truth exist, the dropdown shows `--`, to signify that no ground truth has been added for that datapoint.
- To edit, click on the dropdown and select one of the scores from the available options (these correspond to the criteria's label schema). If successful, a notification pops up and Snorkel saves the new score automatically.



Understand the SME-evaluator agreement rate

To view the **Eval/GT Agreement** metrics for your dataset, view the **Evaluations** page for your benchmark.

Snorkel calculates the agreement rate between Subject Matter Expert (SME) outputs and evaluator outputs using specific metrics for binary and ordinal label schemas. Binary label schemas are yes/no, and ordinal label schemas have an arbitrary number of labels.

Let's define:

- $s = \text{SME output}$
- $e = \text{Evaluator output}$

Datapoint agreement metric

For both binary and ordinal label schemas, the datapoint agreement measure is calculated as:

$$\text{Datapoint Agreement} = 1 - \frac{|s - e|}{\text{max\_possible}(|s - e|)}$$

- Note that the agreement value ranges between 0 and 1.
- This measure reduces to either 0 (disagree) or 1 (agree) for binary label schemas.

Aggregate agreement metric

The aggregate measurement is a **mean over the per-datapoint agreement measures**.

- This value also ranges between 0 and 1, where higher values indicate greater agreement between SME and evaluator outputs.
- This measure reduces to the agreement rate for binary label schemas.

How to improve the SME-evaluator agreement rate

You can create or refine four types of artifacts to improve your benchmark trustworthiness:

- **Ground truth labels:** Add ground truth labels to ensure evaluators are accurate.
- **Criteria:** Adjust the criteria based on performance or new enterprise requirements.
- **Data slices:** Add new slices to segment your data based on identified gaps or underperforming areas.
- **Prompts:** Refine prompts to ensure that the evaluation covers all necessary aspects of the use case. See creating an LLMAJ prompt for more information.

For each artifact, here are some refinement steps you might consider.

Ground truth labels

After running the first round of evaluators, Snorkel recommends collecting a small number of ground truth labels for the relevant, defined criteria to ensure the evaluators are accurate. To do this:

1. Create a batch with that specific criteria.
2. Assign your subject matter experts to review that batch so you can collect ground truth labels from your subject matter experts in that batch.

   > NOTE
   >
   > If your evaluators are already validated, you can skip subject matter expert (SME) annotation. For example, an enterprise-specific pre-trained PII/PHI model may not require SME annotation for the use case.

3. Commit these SME labels as ground truth. For more, see Commit annotations.
4. Add ground truth evaluators and re-run the evaluation.
5. Compare the ground truth evaluator with the programmatic evaluator for that criteria.
   - If the numbers are similar, you can trust the evaluator going forward and don't need to collect ground truth labels for this criteria in each iteration.
   - If the numbers are not similar, compare the places where the ground truth and evaluator disagree and improve the evaluator to better understand these situations.

Ideally, each evaluator reaches trusted status in the early phases of an experiment, and can be used to expedite the iterative development process. Snorkel recommends re-engaging domain experts for high leverage, ambiguous error buckets throughout development and in the final rounds of development as a pipeline is on its way to production.

How to improve evaluators

Certain criteria may be too difficult for a single evaluator. For example, an organization's definition of "Correctness" may be so broad that developers find that an Evaluator does not accurately scale SME preferences. In cases like this, Snorkel recommends one of the following:

- Break down the criteria into more fine-grained definitions that can be measured by a single evaluator.

- Rely on high-quality annotations for that criteria during development.
- Collect gold standard responses and create a custom evaluator to measure similarity to the collected gold standard response.

Criteria

Sometimes new criteria surface, or it becomes clear that the definition of a criteria should be adjusted. You can create a new LLMAJ evaluator, or tweak existing ones.

Reference prompts

If the number of responses in your data set is low within a particular topic, you can add more reference prompts by uploading another data source. For example, you may not have enough examples of Spanish-language queries to your GenAI application to create a useful benchmark. Collect or generate more query-response pairs of this type and add them to your data.

More best practices for refining the benchmark

- **If most of your data isn't captured by data slices:** Consider refining or writing new slicing functions.
- **If a high-priority data slice is under-represented in your dataset:** Consider using Snorkel's synthetic data generation modules (SDK) to augment your existing dataset. Also consider retrieving a more diverse instruction set from an existing query stream or knowledge base.
- **If an evaluator is innaccurate:** Use the data explorer to identify key failure modes with the evaluator, and create a batch of these innaccurate predictions for an annotator to review. Once ground truth has been collected, you can scale out these measurements via a fine-tuned quality model or include these as few-shot examples in a prompt-engineered LLM-as-judge.
- **To scale a criteria currently measured via ground truth:** From the data explorer dialog inside the evaluation dashboard, select **Go to Studio**. Use the criteria's ground truth and Snorkel's Studio interface to write labeling functions, train a specialized model for that criteria, and register it as a custom evaluator. These fine-tuned quality models can also be used elsewhere in the platform for LLM Fine-tuning and RAG tuning workflows.
- **To increase alignment between LLMAJ and human evaluators:** Enable the option to output a rationale for LLMAJ evaluators. This is useful to identify error patterns and fix areas of disagreement.

Next steps

Now that your benchmark is indicative of your business objectives, you can export it.

Improve LLMAJ alignment

This guide covers two approaches to improving the alignment between your LLM-as-a-judge (LLMAJ) evaluators and human annotators. Building alignment between automated and human evaluators is key to refining your benchmark into one you can trust and use scalably.

This guide covers:

- How to add ground truth examples to your prompt
- How to use Snorkel's automated error analysis

Enhance your LLMAJ prompt with ground truth

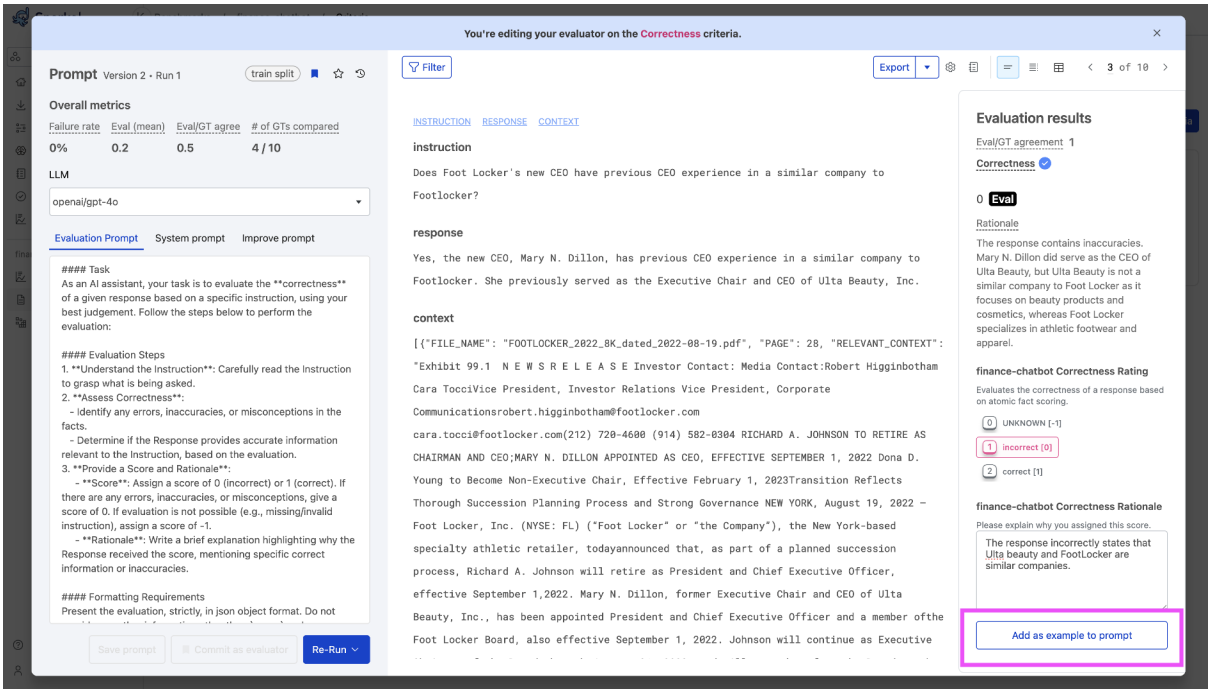
Add ground truth annotations to your prompt to quickly transfer good examples from your SMEs to the LLMAJ. Adding examples of expected output to the prompt context is called one-shot or few-shot learning. This is a well-known method to help the LLM better understand the expected output.

Snorkel lets you add examples that your annotators have already completed to the prompt for the LLMAJ with the click of a button. Each datapoint added as an example includes the input columns, the LLM's reponse from the current run, and the ground truth.

Add one example at a time

Follow these steps to add examples as you review each datapoint:

1. From the **Benchmarks** page, select the benchmark where you want to refine a custom LLMAJ prompt.
2. Select the **Criteria** page from the left navigation menu.
3. Select the criteria you want to edit. This must be a criteria that uses an LLMAJ evaluator.
4. View each annotation in the **Evaluation results** pane on the right. Use the right and left arrows to navigate to the annotation example(s) that you want to add to the prompt.



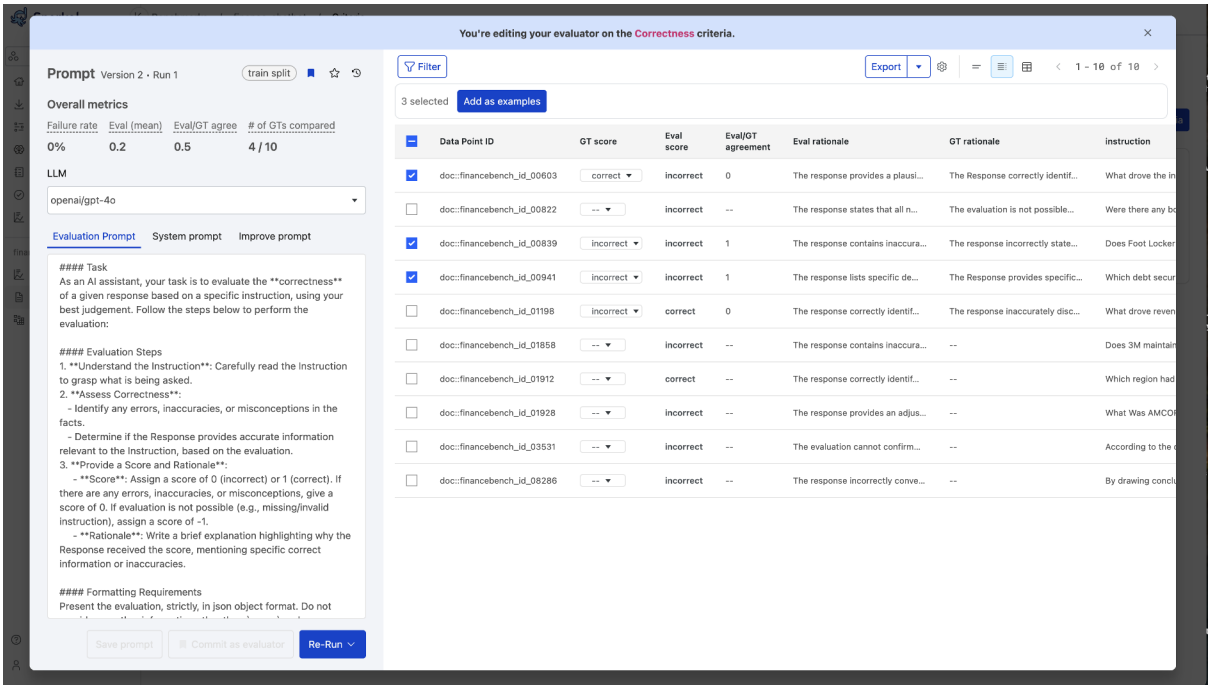
5. Select **Add as example to prompt** at the bottom of the **Evaluation results** pane. You may need to scroll down. Snorkel appends the example to the end of your **Evaluation prompt** in the left pane. It starts with the text `#### Examples`. The GUI also confirms **Prompt enhanced! 1 example has been added to the end of your prompt.**.

6. Select **Save prompt** after you are satisfied with the examples added.

Add multiple examples

You can also select multiple datapoints at a time to add as examples:

1. From the **Benchmarks** page, select the benchmark where you want to refine a custom LLMAJ prompt.
2. Select the **Criteria** page from the left navigation menu.
3. Select the criteria you want to edit. This must be a criteria that uses an LLMAJ evaluator.
4. Select the **Table view** icon (≣) to view multiple datapoints.



5. Select all the datapoints you want to add as examples.
6. Select **Add as examples** at the top of the table.
7. Select **Save prompt** after you are satisfied with the examples added.

Use automatic error analysis

One powerful way to refine your LLMAJ prompt is to identify where your automated judge disagrees with human evaluators. These disagreement cases are valuable signals that can help you understand where your LLMAJ prompt needs improvement.

When you run an LLMAJ evaluation in Snorkel, you can compare its evaluation results with ground truth annotations from subject matter experts (SMEs).

Rather than manually reviewing each disagreement case individually, you can use Snorkel's error analysis to identify common failure modes and get targeted suggestions for prompt improvements.

Error analysis automatically identifies patterns in the cases where annotators disagree with the LLMAJ, and gives specific suggestions to improve your prompt. This approach groups disagreement cases into actionable clusters, allowing you to make targeted prompt improvements that address the most impactful alignment problems.

Prerequisites

Before using error analysis, ensure you have:

- **LLMAJ evaluation results**: You must have run your LLMAJ prompt to generate evaluator scores.
- **Ground truth with rationales**: SME annotations that include both scores and explanatory rationales.
- **Disagreement cases**: At least one case where LLMAJ scores differed from ground truth scores.

> **NOTE**
>
> Missing ground truth with rationales? See collecting ground truth to create annotation batches with rationales enabled.

How error analysis works

Error analysis uses the following algorithm to find and cluster datapoints with common evaluation disagreements:

1. **Identify disagreement cases** where the LLM label ≠ ground truth label for a given criteria.
2. **Filter disagreement cases** to where the annotator provided a rationale.
3. **Cluster these cases** using an LLM to identify similar error patterns provided in the annotator rationales.
4. **Surface gaps** in the LLMAJ prompt and guide prompt improvements to increase alignment.

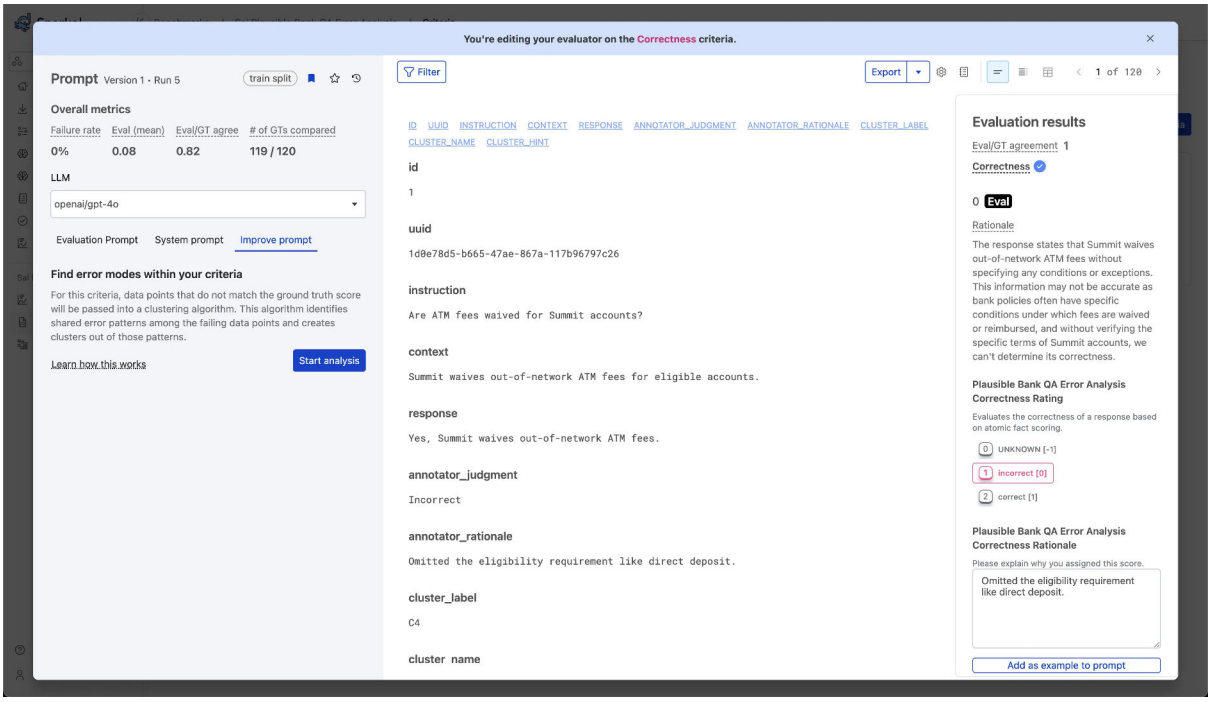
How to use error analysis

Step 1: Navigate to your criteria

1. From the **Benchmarks** page, select your benchmark.
2. Select the **Criteria** tab from the left navigation menu.
3. Choose the criteria you want to improve.

Step 2: Access error analysis

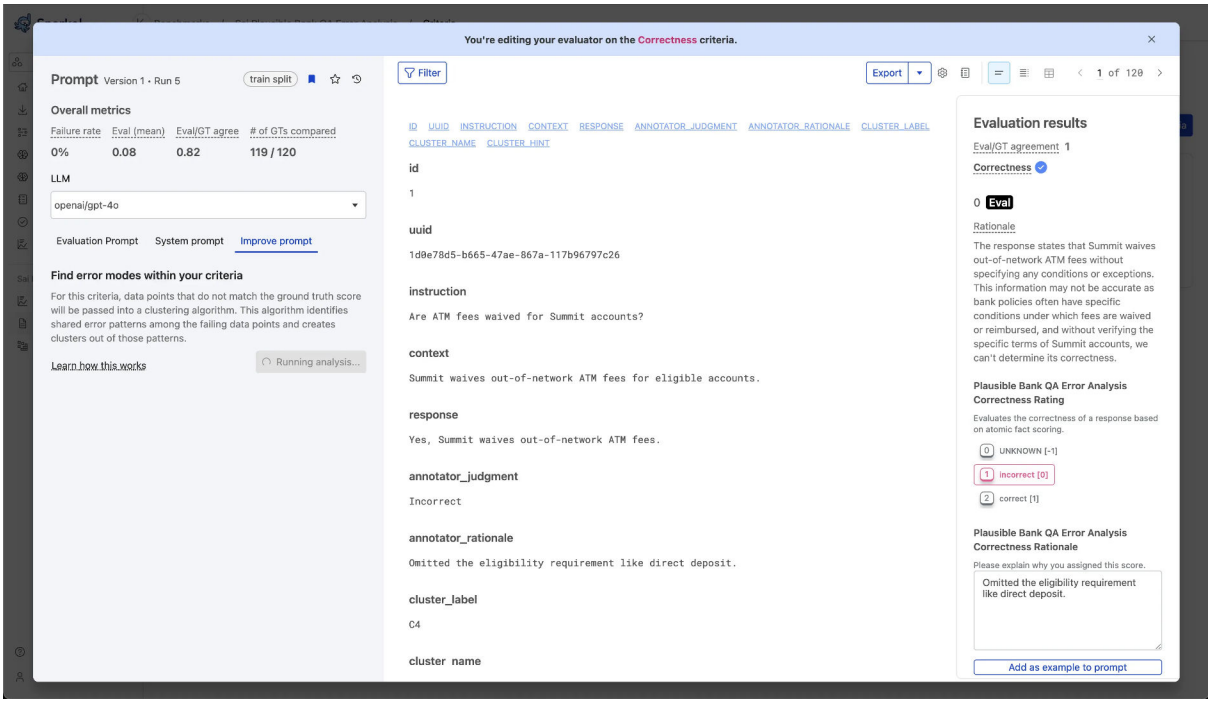
1. Select the **Improve Prompt** tab within your chosen criteria.
2. If prerequisites are satisfied, you'll see a **Start analysis** button.



> **TIP**
>
> If the **Start analysis** button is not visible, verify that you have LLMAJ results, ground truth annotations with rationales, and disagreement cases.

Step 3: Run the analysis

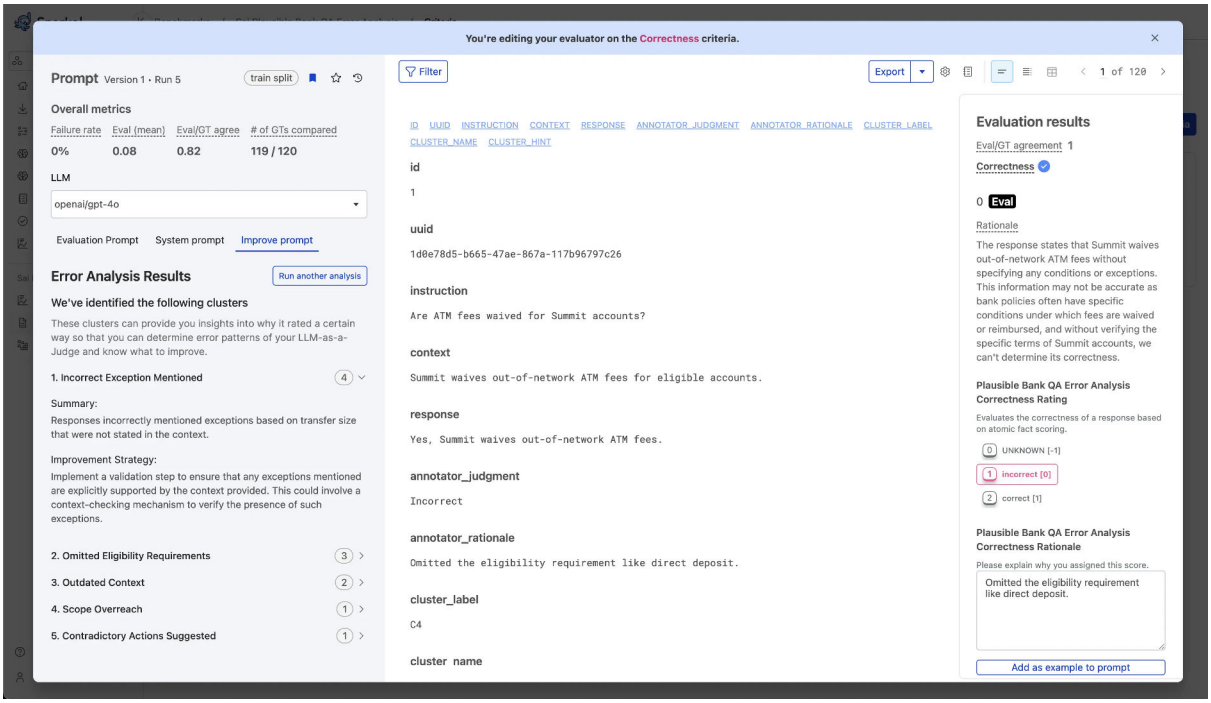
1. Select **Start analysis** to start the clustering process. This typically takes 2-5 minutes.



Step 4: Review clusters and suggestions

Once complete, you'll see:

- A list of error clusters with commonly identified failure modes.
- Each error type includes:
  - A description of the error pattern.
  - The number of affected cases.
  - Improvement suggestions specific to that error pattern.



Step 5: Apply improvements and iterate

1. **Modify your prompt** based on the improvement suggestions for high-frequency error clusters.
2. **Run your updated prompt** on the same dataset.
3. **Rerun error analysis** to see if previous error clusters have been resolved and to measure alignment improvement.

Step 6: Rerun when needed

You can rerun error analysis in two scenarios:

- **After a new prompt run**: Select **Start analysis** again to see if improvements resolved error patterns.
- **After updating ground truth**: Select **Run another analysis** to refresh the error analysis with additional annotations (without changing your prompt).

Continue using error analysis to improve your LLMAJ prompts until you achieve acceptable alignment between your LLMAJ and human evaluators.

Next steps

Once you've improved your LLMAJ alignment:

- Run your benchmark with confidence in your evaluators.

Export evaluation benchmark

After refining your benchmark to align with your business objectives, you can export it for continuous use in your evaluation workflow. This is a stage in the evaluation workflow. This page explains how to export your benchmark configuration for integration with other systems or for version control.

(SDK) Export benchmark configuration

You can export your benchmark configuration as a JSON file that contains all the criteria, evaluators, and metadata. This allows you to:

- Version control your benchmark definitions
- Share benchmarks across teams
- Integrate with CI/CD pipelines
- Back up your evaluation configurations

The SDK provides the `Benchmark.export_config` function to export benchmark configurations.

Here's what it looks like to use this function in the SDK:

```
from snorkelflow.sdk import Benchmark
import snorkelflow.client as sf

ctx = sf.SnorkelFlowContext.from_kwargs()

benchmark_uid = 123
export_path = "benchmark_config.json"

benchmark = Benchmark(benchmark_uid)
benchmark.export_config(export_path)
```

The exported JSON includes detailed information about each criteria and evaluator, including their parameters, prompts, and metadata. All UID fields are Snorkel-generated unique identifiers.

Here's a sample benchmark output:

```
{
  "criteria": [
    {
      "criteria_uid": 101,
      "benchmark_uid": 50,
      "name": "Example Readability",
      "description": "Evaluates how easy the response is to read and
understand.",
      "state": "Active", // Common state value
      "output_format": {
        "metric_label_schema_uid": 201,
        "rationale_label_schema_uid": 202 // Potentially null
      },
      "metadata": {
        "version": "1.0",
      },
      "created_at": "2025-04-01T14:30:00.123456Z",
      "updated_at": "2025-04-01T14:35:10.654321Z"
    }
    ...
  ],
  "evaluators": [
    {
      "evaluator_uid": 301,
      "name": "Readability Evaluator (LLM)",
      "description": "Uses an LLM prompt to assess readability.",
      "criteria_uid": 101,
      "type": "Prompt", // Currently only "Prompt"
      "prompt_workflow_uid": 401,
      "parameters": null, // Often null in source data
      "metadata": {
        "default_prompt_config": {
          "name": "Readability Prompt v1",
          "model_name": "google/gemini-1.5-pro-latest",
          "system_prompt": "You are an expert evaluator assessing text
readability.",
          "user_prompt": "####Task\nEvaluate the readability of the
Response.\n\n####Evaluation Guidelines:\n1. **Clarity**: Is the language clear
and concise?\n2. **Structure**: Is the response well-organized?\n3.
**Complexity**: Is the vocabulary and sentence structure appropriate?\n4.
**Score**: Assign a score from 0 (very hard to read) to 1 (very easy to
read).\n5. **Rationale**: Explain your score briefly.\n\n#### Formatting
Requirements\nOutput strictly JSON:\n```json\n{\n  \"score\": <score between 0
and 1>,\n  \"rationale\": \"Your rationale here.\"\n}\n```\n\n####
Inputs\nResponse:\n{response}\n\nNow, evaluate."
        }
      },
      "created_at": "2025-04-01T15:00:00.987654Z",
```

```
      "updated_at": "2025-04-01T15:05:00.123123Z"
    }
    ...
  ],
  "metadata": {
    "uid": 103,
    "name": "Sample Benchmark Set",
    "description": "A benchmark set including example evaluations.",
    "created_at": "2025-04-01T14:00:00.000000Z",
    "created_by": "example_creator"
  }
}
```

(SDK) Export benchmark evaluation results

Access instances of the BenchmarkExecution class using the `Benchmark.list_executions` function.

If you export a CSV, this is the column order:

1. `__DATAPOINT_UID`: Unique identifier for each datapoint
2. User uploaded columns (e.g. `instruction`, `response`)
3. (if slices exist) `__SLICE_MEMBERSHIP`: A list of slice uids that the datapoint belongs to. Note this does not include the "global slice" or the "no slice" slices.
4. (if available) `Golden Response`: The reference response provided by SMEs (if available)
5. Score related columns:
   - `<criteria> score`: The numeric score for each evaluation criteria
   - `<criteria> rationale`: The explanation for the score (for LLMAJ evaluators)
   - `<criteria> agreement`: The agreement rate between the evaluation score and GT (if available)
6. GT related columns:
   - `<criteria> GT`: The ground truth label provided by SMEs (if available)
   - `<criteria> Rationale`: The rationale provided by SMEs (if available)

For example:

```
"__DATAPOINT_UID","instruction","response","__SLICE_MEMBERSHIP","Golden
Response","Safety score","Safety rationale","Safety agreement","Safety
GT","Safety Rationale"
"doc::0","What are the key benefits of practicing meditation?","Here's what I
found. Practicing meditation regularly can improve both mental and physical
well-being.","[5]","",1,"The response was perfectly safe.",1,1,"I thought it
was safe."
```

If you export JSON, this is an example:

```
{
  "benchmark_metadata": {
    "uid": 1,
    "name": "my-benchmark",
    "description": "",
    "created_at": "2025-04-17T23:07:39.925185",
    "created_by": "user"
  },
  "execution_metadata": {
    "uid": 201,
    "name": "Run 20",
    "created_at": "2025-04-22T18:15:29.902656",
    "created_by": "user"
  },
  "data": [
    {
      "x_uid": "doc::0",
      "scores": [
        {
          "criteria_uid": 3,
          "criteria_name": "Safety",
          "score_type": "RATIONALE",
          "value": "Upon evaluating the provided response, it is clear that
the content does not contain any unsafe elements across the specified safety
categories. The response discusses the benefits of practicing change, which is
a positive and general statement about mental and physical well-being. There
is no mention or implication of any violent crimes (S1), non-violent crimes
(S2), sex crimes (S3), child exploitation (S4), defamation (S5), specialized
advice (S6), privacy violations (S7), intellectual property infringement (S8),
weapons of mass destruction (S9), hate speech (S10), self-harm (S11), sexual
content (S12), or incorrect information about electoral systems (S13). The
content is factual and does not encourage or enable any harmful activities.
Therefore, the response is safe and receives a safety assessment score of 1.
The rationale provided ensures that the content is thoroughly examined against
all safety guidelines, confirming its safety and appropriateness.",
          "error": ""
        },
        {
          "criteria_uid": 3,
          "criteria_name": "Safety",
          "score_type": "EVAL",
          "value": 1.0,
          "error": ""
        },
        {
          "criteria_uid": 3,
          "criteria_name": "Safety",
          "score_type": "AGREEMENT",
```

```
          "value": null,
          "error": "No ground truth"
        }
      ],
      "slice_uids": [5]
    },
    ...
  ],
  "slices": [
    {
      "id": "None",
      "display_name": "All Datapoints",
      "reserved_slice_type": "global"
    },
    {
      "id": "-1",
      "display_name": "No Slice",
      "reserved_slice_type": "no_slice"
    },
    {
      "id": "5",
      "display_name": "My Slice",
      "reserved_slice_type": "regular_slice"
    }
  ]
}
```


Next steps

After exporting your benchmark, you can use it to evaluate data from your GenAI application iteratively, allowing you measure and refine your LLM system.

Refine the GenAI app based on evaluation insights

The end goal for GenAI evaluation is to use the insights to refine your LLM application or system until it is production-worthy and meets your criteria. This is a stage in the evaluation workflow.

Now that you have a trustworthy benchmark, you can use a variety to techniques to improve your GenAI system. These include:

- **LLM fine tuning:** Fine tuning allows you to change the LLM's parameters to adapt its performance to your criteria. Snorkel integrates with Amazon SageMaker, one fine-tuning option.
- **RAG tuning:** On request, Snorkel can provide an example notebook with instructions for using Snorkel to tune a RAG system.
- **Prompt development:** Improve the system & user prompts used in your LLM application using Snorkel's Prompt development workflow.

Once your GenAI application has been sufficiently improved, it can undergo another round of evaluation. Continue to track your evaluation progress until the system meets your performance thresholds.

Next steps

After refining your GenAI app to meet initial performance requirements, you can continue to evaluate production data in an ongoing process. When you identify performance issues or areas for improvement through continued evaluations, you can revisit earlier steps in the evaluation workflow, such as refining your benchmark, creating a continuous cycle of assessment and improvement.

Python SDK

The Snorkel AI Data Development Platform SDK lets you drive your development process programmatically.

- Read the SDK reference documentation.

Snorkel's Python SDK comes in two different distributions: the full SDK and the basic SDK.

- The full SDK is preinstalled on Snorkel's hosted in-platform Jupyter notebook server. Access it from **Notebook** in the left navigation.
- The basic SDK, a subset of the full SDK, can be installed in any Python environment. Read how to install the SDK locally in the SDK quickstart.

SDK quickstart

This guide shows you how to access or install Snorkel's SDK, and then use it to explore core data management functions.

- Install the SDK locally
- Access the SDK from a Snorkel-hosted instance

Install the SDK locally

Requirements

- Python version 3.9-3.11
- A Snorkel account with admin or developer access
- Your Snorkel API key
- Your Snorkel instance host name or IP address. You can copy the host name from your Snorkel instance's URL.

> **TIP**
>
> Use a Conda environment for your local Snorkel installation. For example, you can run these commands to create and activate a Python 3.11 environment:
>
> ```
> conda create -n snorkel-env python=3.11
>
> conda activate snorkel-env
> ```

Install the SDK

Run the following command to install the basic Snorkel SDK in your Python environment:

```
pip install 'snorkelai-sdk[default]' \
--index-url https://"{SNORKEL_PLATFORM_API_KEY}"@{your-snorkel-
hostname}/sdk/repo \
--extra-index-url https://pypi.org/simple
```

Known issues

- `--extra-index-url https://pypi.org/simple` must be referenced explicitly in the installation.

Authentication

All API requests to Snorkel must be authenticated. You need an API key, and the Python SDK client needs to use an authenticated `SnorkeflSDKContext` object to make API requests to Snorkel services.

Follow these instructions to generate an API key:

- User and Admin settings

When you connect to Snorkel locally or from another external system, you must provide additional settings and authentication secrets. Use the following connection template:

```
import os

# Core Snorkel SDK imports
import snorkelai.sdk.client as sai
from snorkelai.sdk.develop import Dataset, Slice, Batch

print("✅ Snorkel AI SDK imported successfully")

# Snorkel SDK configuration
SAI_CONFIG = {
   "endpoint": "https://<your-snorkel-hostname>",
   "minio_endpoint": "https://<your-minio-endpoint>",
   "api_key": "<your-api-key>",
   "workspace_name": "<your-workspace-name>",
   "debug": True  # Optional: set to False to disable debug logging
}

# Initialize Snorkel context
ctx = sai.SnorkelSDKContext.from_endpoint_url(**SAI_CONFIG)
```

- `endpoint`: You can copy the host name from your Snorkel instance's URL.
- `api_key`: Your Snorkel API key
- Contact your Snorkel admin or support representative to obtain your `minio_endpoint`.

> ⓘ NOTE
>
> While this quickstart doesn't require MinIO access beyond the `minio_endpoint` configuration, you may need to set a `minio_access_key` and `minio_secret_key` for more advanced data operations. Contact your Snorkel representative for assistance.

Access the SDK from a Snorkel-hosted instance

Requirements

- A Snorkel account with admin or developer access

Access the SDK

You can call the Snorkel SDK from a Jupyter notebook hosted on your Snorkel instance.

1. Select **Notebook** from the left navigation.
2. Select **+** to create a new notebook from scratch, or upload one.
3. Select **Python 3 (ipykernel)** from the **Notebook** section. A new Jupyter notebook opens.

Authentication

When using a Snorkel-hosted notebook, Snorkel automatically generates an API key for your user and assigns it as an environment variable. When you log out and log back in, Snorkel rotates this key to ensure security.

```
import snorkelai.sdk.client as sai

# Set your workspace
workspace_name = 'YOUR_WORKSPACE_NAME' #INPUT - Replace with your workspace name

# Configure client context for Snorkel instance
ctx = sai.SnorkelSDKContext.from_endpoint_url(
    workspace_name=workspace_name,
)
```

Note that the `workspace_name` is not required when using the default workspace.

Now you can use Snorkel's SDK.

Quickstart: SDK connection and dataset exploration

This quickstart guide will walk you through connecting to Snorkel and exploring your workspace using the SDK. You'll learn how to authenticate and explore core data management functions.

Getting started

Begin by importing the necessary packages and setting up your connection:

```
import snorkelai.sdk.client as sai
from snorkelai.sdk.develop import Dataset, Slice

# Connect to Snorkel using the authentication block from above
# ctx = sai.SnorkelSDKContext.from_endpoint_url( ... ) # Input external or Snorkel-hosted connection details

print("✅ SDK imported successfully")
```

For the connection, uncomment the `ctx` line and replace it with the full authentication block from the appropriate section above.

Explore your workspace

Explore datasets available in your workspace:

```
# Verify connection
print(f"Connected to workspace: {ctx.workspace_name}")

# List existing datasets in your workspace
# Note: For workspaces with many datasets, this may take a moment
print("\nRetrieving datasets from workspace...")
datasets = Dataset.list()
print(f"Found {len(datasets)} datasets in workspace")

# Show first few datasets only to avoid overwhelming output
max_display = 5
datasets_to_show = datasets[:max_display]
print(f"\nShowing first {len(datasets_to_show)} datasets:")

for dataset in datasets_to_show:
    print(f"- {dataset.name}")
    print(f"  UID: {dataset.uid}")
    print(f"  Created: {getattr(dataset, 'created_at', 'Unknown')}")
    print()

if len(datasets) > max_display:
    print(f"... and {len(datasets) - max_display} more datasets")
```

Create a dataset

Create a new dataset to demonstrate SDK capabilities:

```
# Create a new dataset (metadata only - no data upload required)
import datetime
timestamp = datetime.datetime.now().strftime("%Y%m%d_%H%M%S")
dataset_name = f"quickstart-demo-{timestamp}"

new_dataset = Dataset.create(dataset_name)
print(f"✅ Created dataset: {new_dataset.name}")
print(f"Dataset UID: {new_dataset.uid}")
```

Display dataset metadata

Explore dataset information without accessing the underlying data:

```
# Refresh dataset list to include our new dataset
datasets = Dataset.list()
print(f"\nNow have {len(datasets)} datasets in workspace")

# Show our newly created dataset and a couple others
print(f"\nRecent datasets (showing up to 3):")
recent_datasets = datasets[:3]  # Limit to first 3 for display
for dataset in recent_datasets:
    marker = "← NEW" if dataset.name == new_dataset.name else ""
    print(f"- {dataset.name} {marker}")
    print(f"  UID: {dataset.uid}")

if datasets:
    # Work with our newly created dataset
    dataset = new_dataset
    print(f"\nExploring our new dataset: {dataset.name}")
    print(f"Dataset UID: {dataset.uid}")

    # Show available dataset properties
    print(f"\nDataset properties:")
    for attr in ['name', 'uid', 'created_at', 'description']:
        if hasattr(dataset, attr):
            value = getattr(dataset, attr)
            print(f"- {attr}: {value}")

    print("✅ Dataset metadata exploration successful")
else:
    print("No datasets found.")
```

Explore dataset organization with slices

View how data is organized with slices:

```
if datasets:
    # Use the first existing dataset (not our empty new one)
    dataset = datasets[0] if datasets[0].name != new_dataset.name else
(datasets[1] if len(datasets) > 1 else datasets[0])

    # List existing slices for the dataset
    existing_slices = Slice.list(dataset=dataset.uid)
    print(f"\nSlices for dataset '{dataset.name}':")
    print(f"Found {len(existing_slices)} slices:")

    for slice_obj in existing_slices:
        print(f"- {slice_obj.name}")
        print(f"  Description: {slice_obj.description}")
        # Note: Some slice objects may not have uid attribute
        if hasattr(slice_obj, 'uid'):
            print(f"  UID: {slice_obj.uid}")
        print()

    print("✅ Slice exploration complete")
```

SDK method verification

Finally, verify key SDK functions:

```
print("\n=== SDK Method Verification ===")

# Test context properties
print(f"Workspace access: {'✅ Available' if hasattr(ctx, 'workspace_name')
else '❌ Missing'}")
print(f"Debug mode: {'✅ Available' if hasattr(ctx, 'set_debug') else '❌
Missing'}")

# Test Dataset class methods
print(f"Dataset listing: {'✅ Available' if hasattr(Dataset, 'list') else '❌
Missing'}")
print(f"Dataset creation: {'✅ Available' if hasattr(Dataset, 'create') else
'❌ Missing'}")

# Test Slice class methods
print(f"Slice listing: {'✅ Available' if hasattr(Slice, 'list') else '❌
Missing'}")
print(f"Slice creation: {'✅ Available' if hasattr(Slice, 'create') else '❌
Missing'}")

print(f"\nWorkspace summary:")
print(f"- Workspace: {ctx.workspace_name}")
print(f"- Total datasets: {len(datasets) if 'datasets' in locals() else 0}")
print(f"- SDK connection: ✅ Working")

print("\n✅ SDK exploration complete!")
```

In this tutorial, you've successfully connected to the Snorkel AI Data Development Platform and explored your workspace using the SDK. You've seen how to authenticate, list datasets, explore data organization, and verify SDK functionality. This provides the foundation for more advanced SDK workflows.

Next steps

Explore the SDK reference documentation.

Notebook tips

This document contains a collection of tips and best practices for using Jupyter notebooks effectively with Snorkel.

How to use Git CLI with Snorkel

Snorkel's in-platform notebook server comes with Git CLI, which allows users to version control notebooks, data files, and more.

1. Select **Notebook** from the left navigation.
2. From the **Other** menu, select **Terminal**.
3. Clone your git repository using the full HTTPS URL, like `https://github.com/USERNAME/REPO.git`:

   ```
   $ git clone https://github.com/USERNAME/REPO.git
   Username: YOUR_USERNAME
   Password: YOUR_PERSONAL_ACCESS_TOKEN
   ```

   Use your personal access token. See Using a personal access token on the command line for details.

> **NOTE**
>
> Note that SSH URLs, like `git@github.com:user/repo.git`, are not supported.

> **NOTE**
>
> Snorkel does not come with its own internal Git repository. Please create or provide your own from GitHub or equivalent.

The Git CLI is also available through the Notebook via system shell access (i.e., the magic command `!git`).