Evaluation Suite PrPr

Imprint: Mar 28, 2024
Version: v1.0

Summary

Evaluation Suite allows a developer to compute quality metrics for their RAG application without needing to run or deploy the application in RAG Studio.

At a high level, the user journey with Evaluation Suite works as follows:

1. The developer prepares a Delta table in the Unity Catalog with a set of reference questions (and optionally ground-truth for each question). We will call this the **eval set**.
2. The developer runs their existing RAG application through the **eval set** to generate the app's responses for each row. The developer then ETLs these outputs from their application into another Delta table in the Unity Catalog. We will call this the **answer sheet**.
3. The developer invokes the `databricks.rag_eval.evaluate` method on these two tables:

```python
import databricks.rag_eval

rag_eval.evaluate(eval_set_table_name="...", answer_sheet_table_name="...")
```

   Alternatively, if the developer has logged their model using RAG Studio's `log_chain` functionality then they can pass the model directly in lieu of an answer sheet:

```python
import databricks.rag_eval

rag_eval.evaluate(eval_set_table_name="...", model_uri="...")
```

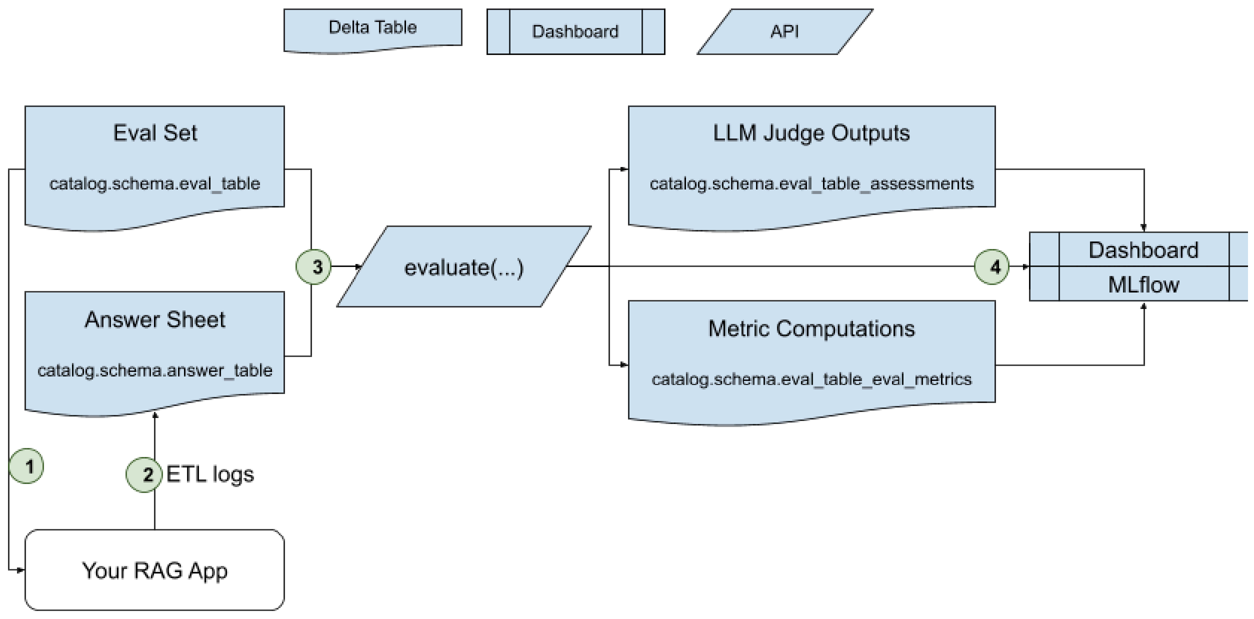
4. The method computes several quality metrics (including LLM-as-a-judge assessments) which are stored in output Delta tables: a table with the computed metrics (called **eval metrics**) and a table with the LLM-as-a-judge assessments (called **assessments**). The method also auto-generates a Lakeview dashboard to visualize these output tables. At this point, the developer can use the dashboard to inspect and discover problematic

inputs, or share the dashboard with other stakeholders. Since the metrics are stored in Delta tables, it is also possible to do a deep dive using SQL or notebooks or any other Lakehouse mechanism for data analysis.

5. After iterating on the RAG application, the developer can generate a new **answer sheet** and then re-invoke the method (the answer sheet includes a column to indicate the version of the application). Evaluation Suite will compute quality metrics for the new answer sheet and ***append*** them to the output tables along with metrics from previous versions. The developer can then peruse the dashboard to compare across versions, or again deep dive into the metric tables as needed.

6. The developer can also iterate on the **eval set** itself. If you update the eval set, and re-run the evaluate command with an **answer sheet**, the output tables (**eval metrics** and **assessments**) will be updated for the version indicated by the answer sheet (e.g., rows of the tuple (request_id, version) will be ***overwritten*** with results based on the new **eval set** & **answer sheet**). The previous results will be stored in the Delta Table's history.

Visualization of the flow:



Limits & future directions

- Judge models are early prototypes and not fully calibrated → please share your feedback on where the models do / don't agree with human raters
- Requires a **single-user cluster with runtime 13.3 or 14.3 (DBR or MLR)** & runs within the cluster → evaluate(..) will be available as an API and thus can be run from local environments or any MLR/DBR version

- MLflow integration → metrics and data are not currently logged to MLflow, but will be in a future version

Creating the input tables

Eval set

The schema of the eval table is as follows:

Column name	Type	Required?	Comment
`request_id`	`STRING`	✅	Id of the request (question)
`request`	`STRING`		A request (question) to the RAG app, e.g., "What is Spark?"
`expected_response`	`STRING`		(Optional) The expected answer to this question
`expected_retrieval_context`	`ARRAY< STRUCT< doc_uri: STRING, content: STRING > >`		(Optional) The expected retrieval context. The entries are ordered in descending rank. Each entry can record the URI of the retrieved doc and optionally the (sub)content that was retrieved.

Note that the table can contain **partial ground truth (**`expected_response` and `expected_retrieval_context`**)** for a request or **no ground-truth at all**.

Answer sheet

The developer needs to create an answer sheet for models that are not developed through RAG Studio. The schema of the answer sheet is as follows:

Column name	Type	Required?	Comment
`request_id`	`STRING`	✅	The identifier for the request. This should match the identifier in the eval set.

Column name	Type	Required?	Comment
`app_version`	`STRING`	✅	The version of the RAG app that was used to generate answers.
`response`	`STRING`	✅	The output of the RAG app on the given question
`retrieval_context`	`ARRAY< STRUCT< doc_uri: SxTRING, content: STRING > >`		(Optional) The retrieved context which was used in the generation of the answer. This has the same structure as the `expected_retrieval_context` in the eval set.

Uploading data into the input tables

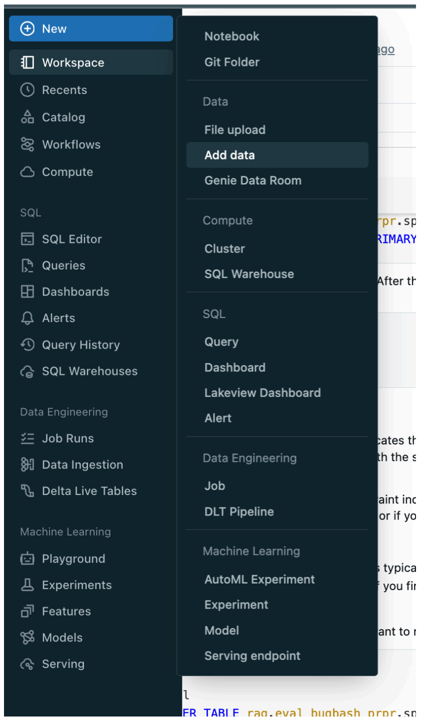
If you have eval data that sits outside of Databricks, you can upload them in Delta tables and transcribe them to the Evaluation Suite using `.jsonl` files. We will walk though an example for the eval set. A similar method can be used for the answer sheet.

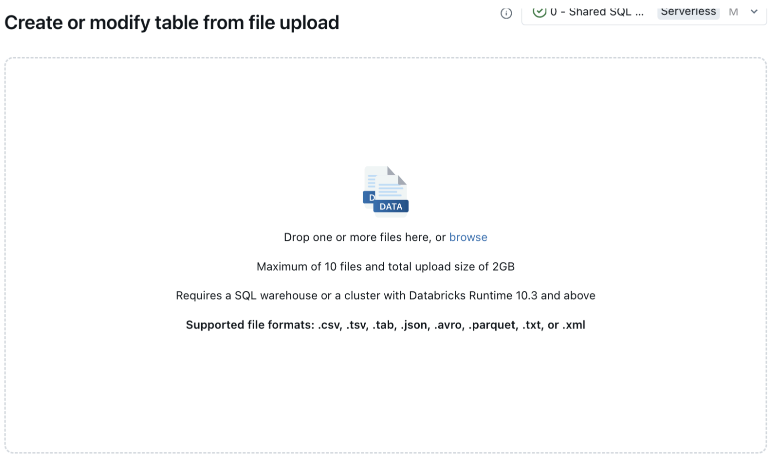
First, create a `.jsonl` file with records that match the eval set schema. Here is an example record:

```
Unset
{ "request_id": "question-1", "request": "What are the main components of Sparks execution
model?", "expected_response": ""Driver, executors, and a cluster manager.",
"expected_retrieval_context": [ { "doc_uri": "one-doc-id", "content": "..."}, { "doc_uri":
"some-other-doc-id", "content": "..."},...] }
```

Next, you can upload the records into a Delta table using any of the following flows:

1. **Import table through the UI.** You can use the add-data UI (aws, azure) to upload your JSONL file to a table. Follow the instructions for “Create or modify table”.




Create or modify table from file upload

Drop one or more files here, or browse

Maximum of 10 files and total upload size of 2GB

Requires a SQL warehouse or a cluster with Databricks Runtime 10.3 and above

Supported file formats: .csv, .tsv, .tab, .json, .avro, .parquet, .txt, or .xml

2. **Call into the read_files SQL method.** Upload first your json file to a UC volume (aws, azure). Then, in the SQL editor or a notebook, you can write the following SQL query to load the data:

```
Unset
SELECT * except (_rescued_data) FROM read_files("<uc volume path to .jsonl
file>", format => "JSON")
```

After you inspect the results and verify that they are correct, you can store them into a table by
embedding the query in a `CREATE TABLE` command, like so:

```
Unset
CREATE TABLE <table_name> AS SELECT * except (_rescued_data) FROM
read_files(...)
```

RAG Evaluation

Getting started **The following uses a staging whl. Substitute with the production whl before sharing externally**

To get started with stand-alone evaluation, first create a **single-user 13.3 or 14.3 LTS** cluster (DBR or MLR). Then attach a notebook and install the whl for the `rag_eval` package.

```
Python
%pip install --quiet "PUT_RAG_EVAL_SUITE_WHEEL_HERE"
```

Restart the python interpreter In a subsequent cell:

```
Python
dbutils.library.restartPython()
```

You are now good to go:

```
Python
from databricks import rag_eval

info = rag_eval.evaluate(eval_set_table_name="uc_path.to.eval_table",
                         answer_sheet_table_name="uc_path.to.answer_sheet")
print(info)

# Command output
Evaluation completed successfully. You can open the dashboard to visualize the results.
You can inspect the updated metrics in
`rag`.`eval_bugbash_prpr`.`docqa_eval_set_eval_metrics`, and the updated assessments in
`rag`.`eval_bugbash_prpr`.`docqa_eval_set_assessments`.

EvaluationOutput(eval_metrics_table_name='`uc_path`.`to`.`eval_table_eval_metrics`',
                  assessments_table_name='`uc_path`.`to`.`eval_table_assessments`',
                  dashboard_id='my_dashboard_id',
                  dashboard_url='url to lakeview dashboard',
                  mlflow_run_id='my_mlflow_run_d',
                  mlflow_run_url='my_mlflow_run_url')
```

As mentioned above, it is possible to evaluate directly a model developed through RAG Studio without going through an answer sheet. As an example, suppose that the model has been logged (through `log_chain`) under the Unity Catalog path `uc_path.to.model` with version 3. Then the evaluate command should be invoked as follows:

```python
from databricks import rag_eval

rag_eval.evaluate(eval_set_table_name="uc_path.to.eval_table",
                  model_uri="models:/uc_path.to.model/3")

# Command output
Evaluation completed successfully. You can open the dashboard to visualize the results.
You can inspect the updated metrics in
`rag`.`eval_bugbash_prpr`.`docqa_eval_set_eval_metrics`, and the updated assessments in
`rag`.`eval_bugbash_prpr`.`docqa_eval_set_assessments`.

EvaluationOutput(eval_metrics_table_name='`uc_path`.`to`.`eval_table_eval_metrics`',
                 assessments_table_name='`uc_path`.`to`.`eval_table_assessments`',
                 dashboard_id='my_dashboard_id',
                 dashboard_url='url to lakeview dashboard',
                 mlflow_run_id='my_mlflow_run_d',
                 mlflow_run_url='my_mlflow_run_url')
```

The two methods (with an answer sheet or with a model) can be used interchangeably on the same evaluation data set, which allows comparing existing models with models developed in RAG studio.

Evaluation Outputs

The call to `rag_eval.evaluate` generates three distinct outputs: **eval metrics table**, **assessments table** and **lakeview dashboard**.

Eval Metrics Table

The eval metrics table is created in the same catalog and schema as the eval table with the suffix `_eval_metrics` (e.g., `c.s.t_eval_metrics` if the input table is named `c.s.t`).

This table has a single row per `request` in the eval set and `app_version` in the answer sheet along with quality metrics computed for the generated outputs and retrieval context. These metrics aim to answer several questions related to the quality of the app, such as:

- Is the retrieval good?
- Are the answers grounded on the retrieved context?
- Are the answers relevant to the question?
- Are the answers correct given some ground-truth?
- Are the answers harmful?
- Which of the answers are the worst in terms of harmfulness, or low relevance/faithfulness/correctness/… ?

Specifically, the following metrics are computed:

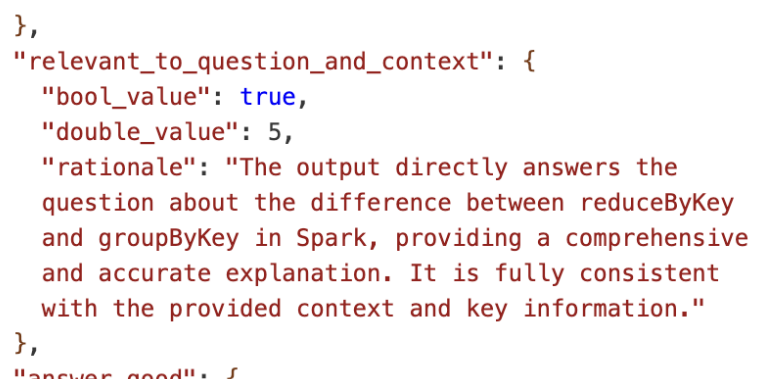
Metric type	**Metric name**	**Explanation**	**Requires ground truth**
Response	`token_count`	Number of tokens in the generated answer	
	`llm_judged_harmful`	Is the answer harmful? (Uses LLM-as-a-judge)	
	`llm_judged_faithful_to_context`	Is the answer faithful to the retrieval context? (Uses LLM-as-a-judge)	
	`llm_judged_relevant_to_question_and_context`	Is the answer relevant given the question and the retrieved context? (Uses LLM-as-a-judge)	
	`llm_judged_relevant_to_question`	Is the answer relevant given the question? (Uses LLM-as-a-judge)	
	`llm_judged_answer_good`	Is the answer good given the question and the ground-truth answer? (Uses LLM-as-a-judge)	✅
Retrieval	`ground_truth_precision_at_k`	Precision@k (k=1,3,5, 10) for the retrieved context	✅
	`ground_truth_recall_at_k`	Recall@k (k=1,3,5, 10) for the retrieved context	✅
	`ground_truth_ndcg_at_k`	NDCG@k (k=1,3,5, 10) for the retrieved context	✅
	`judged_precision_at_k`	Precision@k (k=1,3,5,10) using the LLM-judged context relevance	

The name of the eval-metrics table is included in the `info` struct returned from `rag_eval.evaluate`. You can inspect the contents of the table through a `SELECT * FROM …` query or just by navigating to the Catalog Explorer page of the table and looking at the “Sample Data” tab.

Assessments Table

Several of the metrics in the `_eval_metrics` table rely on LLM judges who generate different assessments (e.g., harmful, relevant, …) for each question and its answer. The assessments table records the outputs of these judges in detail, so that the developer can inspect the assessments and get convinced about their quality and rationale. This table is created in the same catalog and schema as the input eval set with the suffix `_assessments` (e.g., `c.s.t_assessments` if the input table is named `c.s.t`).

The assessments table is keyed on a `request_id` that is either what is used in the input tables or is auto-generated. The column `response_assessment` is a map that is indexed by the name of the metric seen in the eval metrics (“harmful”, “faithful_to_context”, “relevant_to_question_and_context”, …, as mentioned above) and records the results of calling into the LLM judge for this specific metric. As an example, here is an example entry for the “relevant_to_question_and_context” metric:

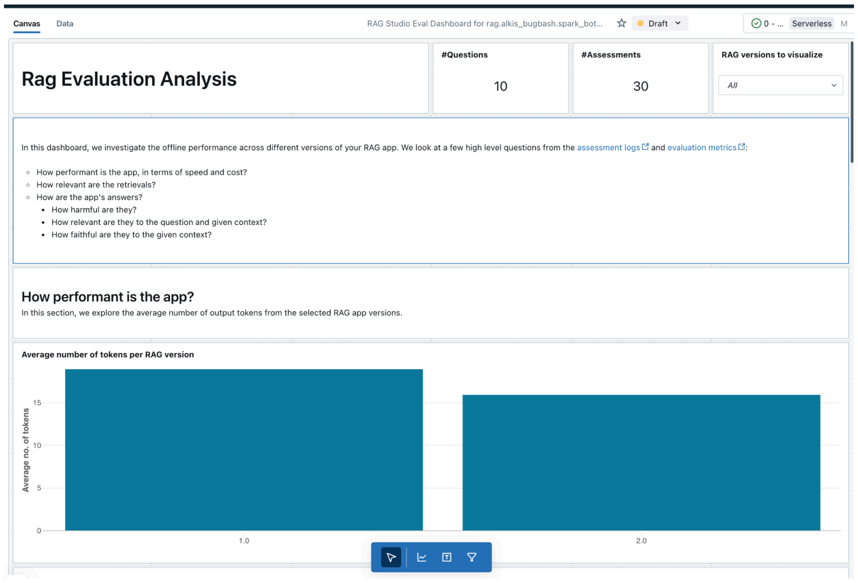


```
},
"relevant_to_question_and_context": {
  "bool_value": true,
  "double_value": 5,
  "rationale": "The output directly answers the
  question about the difference between reduceByKey
  and groupByKey in Spark, providing a comprehensive
  and accurate explanation. It is fully consistent
  with the provided context and key information."
},
"answer_good": {
```

Note that the entry records:
- The boolean value of the assessment, e.g., whether the answer is relevant wrt question and context
- The non-thresholded score generated by the judge (in this example, 5 out of 5)
- The judge’s rationale for giving this rating

Dashboard

The auto-generated dashboard includes several visualizations that help the developer quickly inspect the results and answer some important questions over the computed metrics (e.g., what are the most harmful answers, or which questions have the least faithful answers).


Canvas
Data
RAG Studio Eval Dashboard for rag.alkis_bugbash.spark_bot...
Draft
0 - ...
Serverless
M

Rag Evaluation Analysis

#Questions 10

#Assessments 30

RAG versions to visualize: All

In this dashboard, we investigate the offline performance across different versions of your RAG app. We look at a few high level questions from the assessment logs and evaluation metrics:

- How performant is the app, in terms of speed and cost?
- How relevant are the retrievals?
- How are the app's answers?
  - How harmful are they?
  - How relevant are they to the question and given context?
  - How faithful are they to the given context?

How performant is the app?

In this section, we explore the average number of output tokens from the selected RAG app versions.

The dashboard is fully editable by the developer which means that he/she can change the existing visualizations or add new ones. These changes persist across runs of `rag_eval.evaluate` and enable the developer to create a customized dashboard that streamlines the inspection of the results or facilitates easier sharing of the results with other stakeholders.

Note: Initially the dashboard is created in draft mode which allows for editing and adding visualizations. You can click on “Publish” in order to get a non-editable version of the dashboard that is suitable for sharing with other stakeholders.

Supporting the development cycle

The purpose of `rag_eval.evaluate` is to help the developer determine the next steps for improving the quality of the RAG application. After doing this iteration and creating the next version of the app, the developer will want to re-evaluate and compare the performance of the new version against the old results. This is possible by just calling `rag_eval.evaluate` with the same eval set and the answer sheet from the new version:

```
Unset
info = rag_eval.evaluate(eval_set_table_name="uc_path.to.eval_table",
                         answer_sheet_table_name="uc_path.to.new_answer_sheet")
print(info)
```

Note that each `rag.evaluate()` call for the same eval_set ***updates*** the output tables with the metrics and assessments related to the new answer sheet (which has its own distinct app_version). In other words, the output tables accumulate the results of invoking rag_eval.evaluate on different answer sheets. Accordingly, the dashboard contains a "version" filter which allows the developer to inspect the results for a specific app version, or to compare results across app versions.

Configuring the evaluation

rag_eval.evaluate uses a default config to determine which metrics to compute and which endpoint to use for LLM judging. You can override this behavior through the optional config argument.

The config is encoded as a YAML string with the following default:

```
Unset
assessment_judges:
 - judge_name: databricks_eval
   assessments:
  - harmful
  - faithful_to_context
  - relevant_to_question_and_context
  - answer_good
  -  relevant_to_question
```

At this point you can disable some of the assessments if they are not relevant for your use case. Here is an example of passing a trimmed-down config to rag_eval.evaluate:

```
Unset
# Create a config that only keeps answer_good and harmful
config="""
assessment_judges:
 - judge_name: databricks_eval
   assessments:
   - harmful
   - answer_good
"""

rag_eval.evaluate(eval_set_table_name="...",
                  answer_sheet_table_name="...",
                  config=config)
```

The default config uses the pre-configured Databricks endpoint with Llama-2-70B as the LLM judge. If you do not have access to this endpoint, or if you want to use a different model (e.g., OpenAI or Mixtral for LLM judging, then configure an endpoint with your preferred LLM (documentation) and use that in the config.

Few-shot prompting for the built-in judges

To increase the quality of the judges for your specific use case, you can pass domain-specific examples to the built-in judges by providing a few True/False examples for each type of the assessment. We suggest providing at least 2 True and 2 False examples. The best examples are ones that the judges previously got wrong (e.g.., you provide a corrected response as the example) or challenging examples (e.g., examples that are nuanced and difficult to make a true/false determination for). The examples can be specified as part of the configuration, as follows:

```
Unset
# Example config where we will supply a positive and a negative example for
# the `answer_good` assessment.

config="""
assessment_judges:
 - judge_name: databricks_eval
  assessments:
  - harmful
  - answer_good:
     examples:
       - request: "What is Apache Spark?"
         response: "Spark is what happens when there is fire."
         expected_response: "Spark is a distributed data processing engine."
         value: False
         rationale: "The output is completely incorrect"
       - request: "What is RAG?"
         response: "Retrieval-Augmented-Generation is a powerful paradigm for
using LLMs"
         expected_response: "RAG is retrieval augmented generation"
         value: True
         rationale: "The output matches well the expected response."
  - faithful_to_context:
      examples:
        - request: "What is Apache Spark?"
          response: "Spark is what happens when there is fire."
          context: "Apache Spark is an open-source unified analytics engine for
large-scale data processing."
          value: False
  - context_relevant_to_question:
      examples:
        - request: "What is Apache Spark?"
          context: "Apache Spark is an open-source unified analytics engine for
large-scale data processing."
          value: True
          rationale: "The retrieved co  ntext accurately answers the question."
"""
```

Which fields you provide vary based on the judge. This config provides two examples for the `answer_good` judge, one for the `True` assessment and another one for `False`. The contents of the examples depend on the assessment. In this case, `answer_good` compares the response to an expected response for the same request, so the example needs to provide all three components. The following table outlines which fields are expected in the examples of different judges. In all cases, the values for these fields are strings.

☑️Optional
✅Required

	request	response	context	expected_response	rationale	value
What it is?	*The actual contents of the few-shot example*				*Reasoning for the value*	*True or False*
faithful_to_context	✅	✅	✅ Contents of a single doc/snippet or concatenated contents of several.		☑️	✅
relevant_to_question_and_context	✅	✅	✅ Contents of a single doc/snippet or concatenated contents of several.		☑️	✅
relevant_to_question	✅	✅			☑️	✅
answer_good	✅	✅		✅	☑️	✅
context_relevant_to_question	✅		✅		☑️	✅

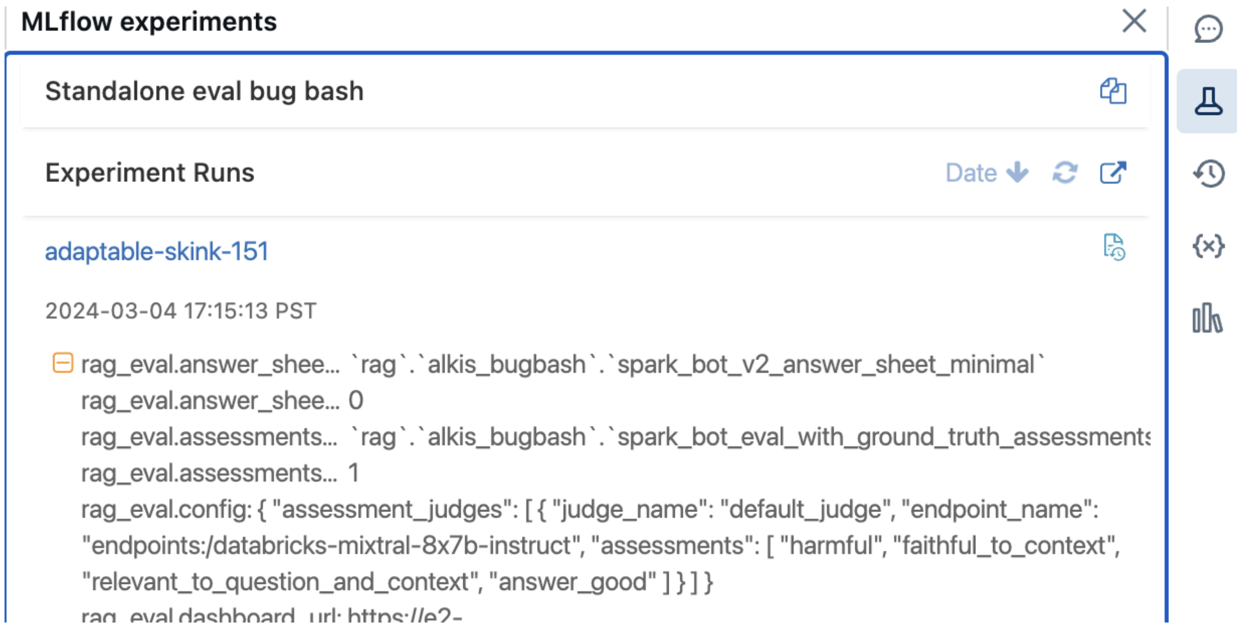
			Contents of a single doc/snippet			

Integration with mlflow

`rag_eval.evaluate` records metadata about the evaluation under the active mlflow run (or, creates a new run if none is active). The metadata includes:

- A URL to the dashboard
- The names of the output tables
- The names of the input tables
- The Delta versions of the input and output tables
- The configuration used

As always, you can access this information by clicking into the experiment tab on the right nav:



```
rag_eval.answer_shee... `rag`.`alkis_bugbash`.`spark_bot_v2_answer_sheet_minimal`
rag_eval.answer_shee... 0
rag_eval.assessments... `rag`.`alkis_bugbash`.`spark_bot_eval_with_ground_truth_assessments
rag_eval.assessments... 1
rag_eval.config: { "assessment_judges": [ { "judge_name": "default_judge", "endpoint_name": "endpoints:/databricks-mixtral-8x7b-instruct", "assessments": [ "harmful", "faithful_to_context", "relevant_to_question_and_context", "answer_good" ] } ] }
rag_eval.dashboard_url: https://e2-
```

This metadata immediately helps with reproducibility, as it allows retrieving the contents of the input tables at the time of the specific evaluation using [Delta's time-travel](). The metadata can also help reason about comparison between versions, e.g., by ensuring that the evaluation configs lead to comparable results.

Important note

LLM Judge outputs are intended to help customers optimize their RAG applications and may not be used to develop or improve any large language models.
LLM Judge may use models available under Foundation Model APIs, among others.

FAQ/Troubleshooting

I messed up the dashboard and I want to reset it

Simply delete the dashboard (Select “Move to trash” from the upper-right drop-down menu). Next time that `rag_eval.evaluate` runs it will re-create it.

I used the wrong config for an evaluation

Simply fix the config and re-run `rag_eval.evaluate`. The metrics for the app version in the answer sheet will be updated in place in the outputs.

I have a question that is not addressed here. Where can I get help?

Please send a message to `rag-feedback@databricks.com` to contact the Databricks team.