Enhancing Duet AI Assistant Training and Dataflow Cookbook repository with new Beam Content

Dear Apache Beam Community,

We are excited to share that we are currently working on a set of golden prompts and responses enhancing the Duet AI training and specifically tailored to improve Dataflow/Apache Beam development experience.

The training material is extensive, covering a wide range of topics from the Apache Beam documentation. We've categorized the into the following sections:

Knowledge Lookup Prompts

These prompts encompass both basic and advanced concepts in Apache Beam with specific focus on 11 IO Transforms:

1. Learning Apache Beam
2. Pipeline
3. Pipeline options
4. PCollection
5. PTransform
6. Schemas
7. Runners
8. Windowing
9. Triggers
10. Metrics
11. State
12. Timers
13. Splittable DoFn
14. Pipeline Patterns
15. Multi-Language Pipelines
16. Pipeline Development Lifecycle
17. AI/ML Pipelines
18. Kafka IO
19. PubSub IO
20. BigQuery IO
21. BigTable IO
22. Spanner IO
23. GCS IO
24. File IO JSON

25. File IO CSV
26. File IO Avro
27. File IO Parquet
28. JDBC IO

Code Generation Prompts

Aimed at assisting developers in generating code for various IO transforms, streamlining the development process.

1. Kafka IO
2. PubSub IO
3. BigQuery IO
4. BigTable IO
5. Spanner IO
6. GCS IO
7. File IO JSON
8. File IO CSV
9. File IO Avro
10. File IO Parquet
11. JDBC IO

Code Explanation Prompts

These prompts are designed to clarify and explain the functioning and usage of different IO transforms within Apache Beam:

1. Kafka IO
2. PubSub IO
3. BigQuery IO
4. BigTable IO
5. Spanner IO
6. GCS IO
7. File IO JSON
8. File IO CSV
9. File IO Avro
10. File IO Parquet
11. JDBC IO

Alongside the Duet AI training enhancements, we are also expanding the Dataflow-Cookbook repository. This update will include additional pipelines that showcase practical implementations and best practices, further enriching the Duet AI experience for Apache Beam developers.

We believe these additions will be a valuable resource for both new and experienced Beam users, streamlining the learning curve and enhancing productivity.

Your feedback and suggestions are invaluable to us. We would love to collaborate and refine these resources further. If you have any ideas, insights, or areas you believe should be included or emphasized, please feel free to share.

Here are examples of prompts for your reference:

Prompt type	Prompt/Response
Knowledge lookup	**Prompt:** How can I learn Apache Beam? **Response:** Learning Apache Beam is a valuable skill for data processing and analysis. Here are some resources to get you started: For a hands-on introduction, you can start by exploring the [WordCount](https://beam.apache.org/get-started/wordcount-example/) - a simple example that demonstrates the basic concepts of Apache Beam. The [Tour of Beam](https://tour.beam.apache.org/) is an interactive tour that allows you to learn how to write Beam code. It provides a sandbox environment to write and run pipelines while exploring different concepts. The [Beam Playground](https://play.beam.apache.org/) offers an interactive space to experiment with Beam's transforms and examples without installing anything on your system. Once you have a basic understanding of Apache Beam, you can start to build your own pipelines. The Apache Beam website has a number of resources that can help you get started, including [Apache Beam Documentation](https://beam.apache.org/documentation/), and [Apache Beam Community](https://beam.apache.org/community/) For those interested in the source code, you can explore the [Apache Beam repository](https://github.com/apache/beam) on GitHub.

	By leveraging these resources, you can build a strong foundation in Apache Beam and start constructing your own data processing pipelines. Additionally, Apache Beam is an open-source project, and contributions are welcome. Whether you're fixing bugs, improving documentation, or adding new features, your work can benefit the entire Beam community. To learn more about how you can contribute, visit the [Contribute](https://beam.apache.org/contribute/#) section of the Apache Beam website.
Code generation	**Prompt:** Write the python code to read messages from a PubSub subscription **Response:** You can read messages from a PubSub subscription or topic using the `ReadFromPubSub` transform. PubSub is currently supported only in streaming pipelines. The following python code reads messages from a PubSub subscription. The subscription is provided as a command line argument. The messages are logged to the console: ```python import logging import apache_beam as beam from apache_beam import Map from apache_beam.io import ReadFromPubSub from apache_beam.options.pipeline_options import PipelineOptions class PubSubReadOptions(PipelineOptions): """ Configure pipeline options for PubSub read transform. """ @classmethod

```
    def _add_argparse_args(cls, parser):
      parser.add_argument(
          "--subscription",
          required=True,
          help="PubSub subscription to read from.")

def read_subscription():
  """Read from PubSub subscription function."""

  #parse pipeline options
  #streaming=True is required for a streaming pipeline
  options = PubSubReadOptions(streaming=True)

  with beam.Pipeline(options=options) as p:
    #this pipeline reads from a PubSub subscription and
logs the messages to the console
    (p | "Read PubSub subscription" >>
ReadFromPubSub(subscription=options.subscription)
       | "Format message" >> Map(lambda message:
f"Received message:\n{message}\n")
       | Map(logging.info))
```

Reading messages directly from a topic is also supported. A temporary subscription will be created automatically.

The messages could be returned as a byte string or as PubsubMessage objects. This behavior is controlled by the `with_attributes` parameter.

See [PubSub IO](https://beam.apache.org/releases/pydoc/current/apache_beam.io.gcp.pubsub.html) transform documentation for more details.

For a common pattern for configuring pipeline options see here [Pipeline option pattern](https://beam.apache.org/documentation/patterns/pipeline-options/).

Code explanation	**Prompt:** What does this code do? ```python pipeline_options = PipelineOptions(pipeline_args) with beam.Pipeline(options=options) as p: (p	"Read" >> ReadFromPubSub(subscription=input_subscription) 	"Process" >> ParDo(ProcessMessage()) 	"Write" >> WriteToPubSub(topic=output_topic)) ``` **Response:** This code reads messages from a [PubSub](https://cloud.google.com/pubsub) subscription `input_subscription` using the `ReadFromPubSub`, processes them and writes the resulting collection to a PubSub topic `output_topic` using the `WriteToPubSub` transform. PubSub is currently supported only in streaming pipelines. Reading messages directly from a topic is also supported. In this case a temporary subscription will be created automatically. `ReadFromPubSub` produces a `PCollection` of `PubsubMessage` objects or a `PCollection` of byte sequences. Behavior is controlled by the `with_attributes` parameter with byte sequences being the default. For more on PCollections see the [Beam Programming Guide](https://beam.apache.org/documentation/basics/#pcollection). Processing of the messages is done by the

	`ProcessMessage` class. This class is a subclass of the `DoFn` class. Simplest implementation of ProcessMesage class could be something like: ```python class ProcessMessage(beam.DoFn): def process(self, element): yield element ``` More on `DoFn` class can be found [here](https://beam.apache.org/documentation/programming-guide/#dofn). See [PubSub IO](https://beam.apache.org/releases/pydoc/current/apache_beam.io.gcp.pubsub.html) transform documentation for more details. For a common pattern for configuring pipeline options see here [Pipeline option pattern](https://beam.apache.org/documentation/patterns/pipeline-options/).