
BEAM
SUMMIT

# Integrating LLMs and Embedding models into Beam pipelines using langchain

Ganesh Sivakumar

BEAM SUMMIT NYC 2025

# Agenda

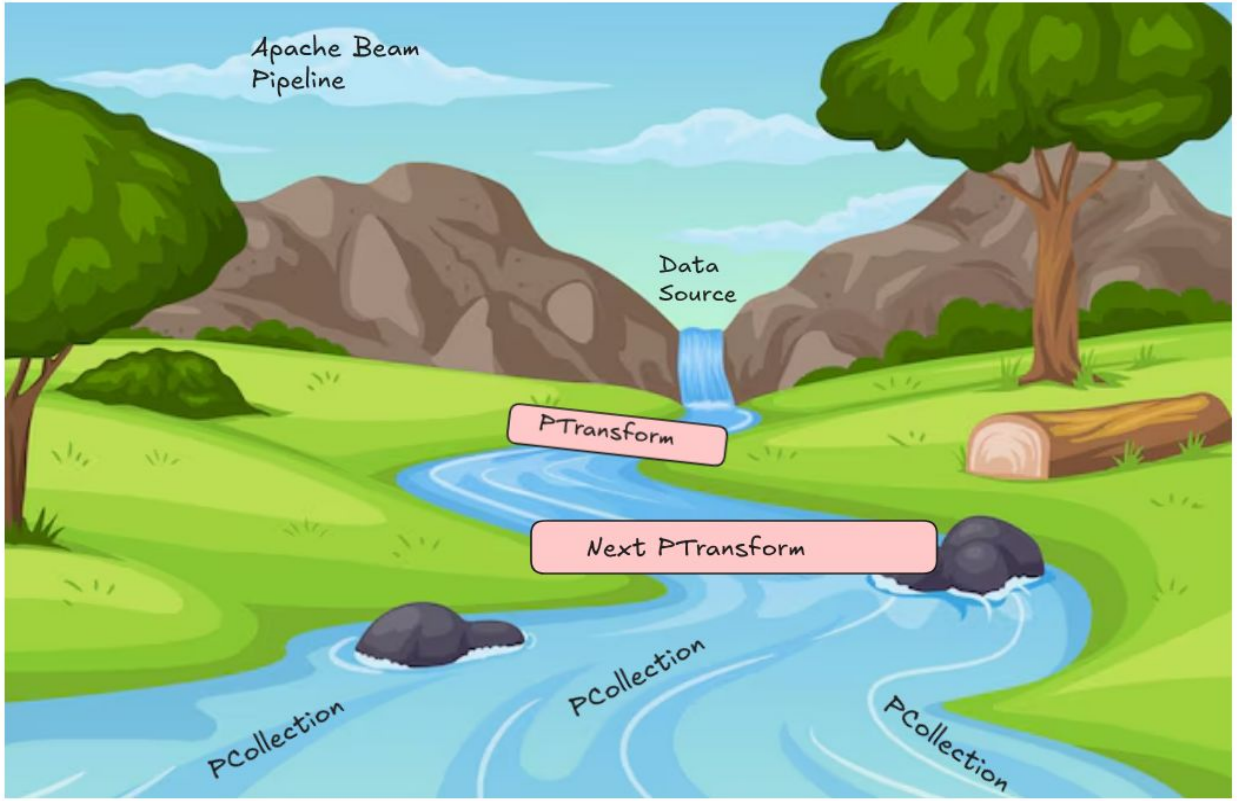
- PTransform
- LangchainBeam: PTransform Overview
- Usage in pipeline
- Demo

# PTransform

A PTransform in Apache Beam is a building block that represents a single stage in a  pipeline. It takes in data, applies some transformation logic—like filtering out invalid records, extracting fields, aggregating data, or detecting anomalies—and produces new transformed data as output

[Output PCollection] = [Input PCollection].apply( Transform )

# Inside a beam pipeline...



# LLM transform in langchain-beam

Performs remote LLM inference in the transform using langchain to leverage the model to process input data..

Combines the apache beam's abstraction with the capabilities of Large Language Models, such as generation, completion, classification, and reasoning to process the data within apache beam pipeline

```
Input:      PCollection<String>
                    ↓
Transform:  LangchainBeam.run()
                    ↓
Output:     PCollection<LangchainBeamOutput>
```

# How to use LLM transform

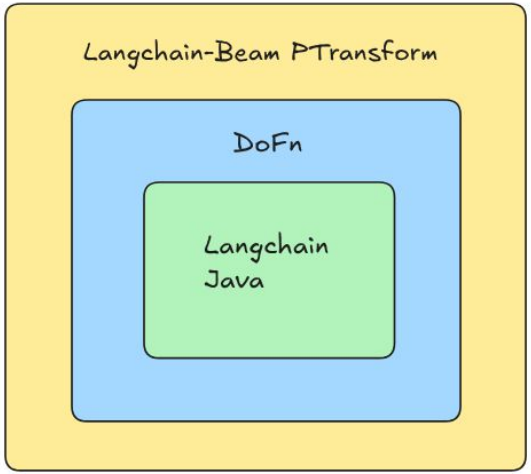
```
// instruction prompt on how to process the input element
String prompt = "Categorize the product review as Positive or Negative.";

// Create model options with the model and its parameters
OpenAiModelOptions modelOptions = OpenAiModelOptions.builder()
        .modelName("gpt-4o-mini")
        .apiKey(apiKey)
        .build();

// create LangchainModelHandler to pass it to LangchainBeam transform
LangchainModelHandler handler = new LangchainModelHandler(modelOption, prompt);

    pipeline
      .apply("ReadInput", ...)
      .apply("llm transform", LangchainBeam.run(handler))
      .apply("WriteOutput", ...);
```

# How does it work

**1. `LangchainBeam` — The PTransform**

- It's the outermost transform applied on a `PCollection<String>` inputs.
- Accepts a `LangchainModelHandler` which contains:
  - Model options, Prompt , Output formatting instructions
- It internally applies a `DoFn` called `LangchainBeamDoFn`

```
LangchainBeam.run(handler)
```



# Transform internals

**2. `LangchainBeamDoFn` — The DoFn**

This is the core processing function responsible for applying the LLM to each element in the pipeline.

**Initialization Phase (`@Setup`)**

- Langchain4j's `ChatLanguageModel` is instantiated using the model options provided via handler

**Per-element Processing (`@ProcessElement`)**

For each input element:

1. A final prompt is dynamically constructed using prompt template, where the raw input and instruction are injected into a templated format.
2. `ChatLanguageModel` invoked with the final prompt.
3. The result is wrapped in a `LangchainBeamOutput` — a data class holding both the input and the model's output

# Prompt template

At the heart of transform is a prompt template — but this isn't just a simple string. This prompt acts like an instruction contract between your pipeline and the language model.

```
public static String PROMPT = """
        You are a PTransform within an Apache Beam pipeline.
        Your objective is to take the input element: {%s}
        and execute the following instruction prompt: {%s}.
        Provide the output that fulfills this instruction and
        the output should be strictly in this format: {%s}
        """;
```

```
@ProcessElement
public void processElement(ProcessContext context) {
    String input = context.element();

    final String finalPrompt = String.format(ModelPrompt.PROMPT, input, handler.getPrompt(),
            modelOutputFormat);

    try {
        modelOutput = model.generate(finalPrompt);
```

# With structured model outputs

```
Map<String, String> outputFormat = new HashMap<>();
outputFormat.put(key:"review", value:"product review");
outputFormat.put(key:"sentiment", value:"sentiment of the product, postive or negative");

LangchainModelHandler handler = new LangchainModelHandler(modelOptions, prompt, outputFormat);
```

```
p.apply(TextIO.read().from("/home/ganesh/Downloads/product_reviews.csv"))
        .apply(LangchainBeam.run(handler))
        .apply(ParDo.of(new DoFn<LangchainBeamOutput, Void>() {

            @ProcessElement
            public void processElement(@Element LangchainBeamOutput out) {
                System.out.println("Model Output: " + out.getOutput());
                // Model Output:
                // {"sentiment":"Positive","review":"Very satisfied with my purchase. Customer
                // support was helpful too."}
            }
        }));

p.run();
```

# Embeddings transform

The transform integrates embedding models into a PTransform and uses the model to generate embedding. Takes a pcollection of text inputs, uses the model to generate vector embeddings and outputs the EmbeddingOutput which warps the input string and its embeddings.

```
Input:     PCollection<String>
                ↓
Transform: LangchainBeamEmbedding.embed(handler)
                ↓
Output:    PCollection<EmbeddingOutput>
```

# Embeddings pipeline

```
// create Embedding model options
OpenAiEmbeddingModelOptions modelOptions = OpenAiEmbeddingModelOptions.builder().apikey(apiKey)
        .modelName("text-embedding-3-small")
        .build();

EmbeddingModelHandler handler = new EmbeddingModelHandler(modelOptions);

// create beam pipeline
Pipeline p = Pipeline.create(options);

// load data
p.apply(TextIO.read().from("/home/ganesh/Downloads/product_reviews.csv"))

        // apply LangchainBeamEmbedding transform to generate embeddings for data
        .apply(LangchainBeamEmbedding.embed(handler))
        .apply(ParDo.of(new DoFn<EmbeddingOutput, Void>() {

            @ProcessElement
            public void processElement(@Element EmbeddingOutput out) {
                System.out.println("Embedding output " + Arrays.toString(out.getEmbedding().vector()));
            }
        }));

// run pipeline
p.run();
```

# Pipeline Demo

Ganesh Sivakumar

# QUESTIONS?

Linkedin:
linkedin.com/in/ganesh-sivakumar

Github:
https://github.com/Ganeshsivakumar

BEAM SUMMIT
NYC 2025