Expanding differentially private solutions: Introducing PipelineDP

Miguel Guevara &
Vadym Doroshenko &
Yurii Sushko

mgt@google.com, dvadym@google.com, sushko@google.com

PEPR 2022

Anonymous data

Anonymized data reduces privacy and security risks.

It can provide comparable statistical insights.

It can be even better, as it removes some of the collection noise (e.g., random outliers).

Differential Privacy

An algorithm is differentially private if the output doesn't change "much" when a single person is added to the database.

Definition 2.4 (Differential Privacy). A randomized algorithm $\mathcal{M}$ with domain $\mathbb{N}^{|\mathcal{X}|}$ is (ε, δ)-differentially private if for all $\mathcal{S} \subseteq \text{Range}(\mathcal{M})$ and for all $x, y \in \mathbb{N}^{|\mathcal{X}|}$ such that $\|x - y\|_1 \leq 1$:

$$\Pr[\mathcal{M}(x) \in \mathcal{S}] \leq \exp(\varepsilon) \Pr[\mathcal{M}(y) \in \mathcal{S}] + \delta,$$

Challenges of differential privacy

Simple conceptually

- Add noise to data
- Formal guarantees

Difficult in practice

- Tricky to implement correctly (like cryptography)
- Lots of implementation subtleties
- Adding noise can make data less useful

PipelineDP

An open source Python framework for differentially private aggregations to large datasets using batch processing systems such as Apache Spark and Apache Beam

Goals of PipelineDP

01

Easy and accessible to non-experts

02

Scalable with support for Apache Spark and Beam

03

Practical to achieve reasonable utility

PipelineDP computes statistical queries

Python equivalent of

```
SELECT WITH DIFFERENTIAL PRIVACY aggregation_function(value)
FROM table
GROUP BY key

where aggregation_function is COUNT, SUM, MEAN, PERCENTILE etc.
```

Example dataset of restaurant visits

visitor_id	restaurant_id	rating
v1	r1	5
v1	r2	4
v2	r3	2
...	...	...

PipelineDP computes statistical queries

Python equivalent of

```
SELECT WITH DIFFERENTIAL PRIVACY ANON_MEAN(rating)
FROM restaraunt_visits
GROUP BY restaraunt_id
```

... or, in plain English, calculate the average rating of each restaurant.

```
restaraunt_visits = … # Load data of restaurants visits
backend = pipeline_dp.LocalBackend() # Run locally


budget_accountant = pipeline_dp.NaiveBudgetAccountant(
                                 total_epsilon=1,
                                 total_delta=1e-6) # Set the budget



# Create DPEngine which will execute the logic
dp_engine = pipeline_dp.DPEngine(budget_accountant, backend)


# Define privacy ID, partition key and value extractors
data_extractors = pipeline_dp.DataExtractors(
  partition_extractor=lambda row: row.restaurant_id, # group by key
  privacy_id_extractor=lambda row: row.user_id,
  value_extractor=lambda row: row.rating) # Value to aggregate
```

```
# Configure the aggregation parameters
params = pipeline_dp.AggregateParams(
  # DP method
  noise_kind=pipeline_dp.NoiseKind.LAPLACE,
  # DP metrics to compute
  metrics=[pipeline_dp.Metrics.MEAN],
  # Limits visits contributed by a visitor
  max_partitions_contributed=3,  # A visitor can contribute up to 3 days
  max_contributions_per_partition=2)  # … and up to 2 visits per day



# Create a computational graph for the aggregation
dp_result = dp_engine.aggregate(restaraunt_visits, params,
   data_extractors)
```

```python
# Assume having running Spark cluster.
restaraunt_visits = … # Load data of restaurants visits with Spark
backend = pipeline_dp.SparkBackend() # Run on Spark cluster

budget_accountant = pipeline_dp.NaiveBudgetAccountant(
                                 total_epsilon=1,
                                 total_delta=1e-6) # Set the budget


# Create DPEngine which will execute the logic
dp_engine = pipeline_dp.DPEngine(budget_accountant, backend)


# Define privacy ID, partition key and value extractors
data_extractors = pipeline_dp.DataExtractors(
  partition_extractor=lambda row: row.restaurant_id, # group by key
  privacy_id_extractor=lambda row: row.user_id,
  value_extractor=lambda row: row.rating) # Value to aggregate
```

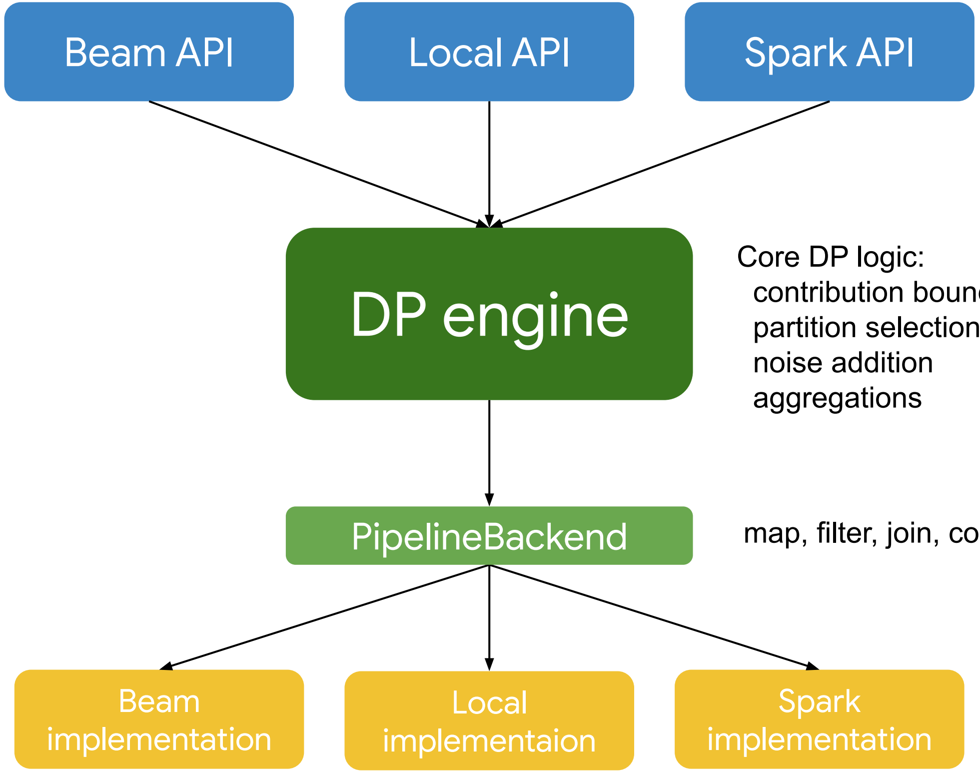
What PipelineDP is and isn't

It performs DP computations and manages a budget per pipeline.

It does not enforce privacy budget usage per analysts, dataset etc.

It does not perform any data access management.

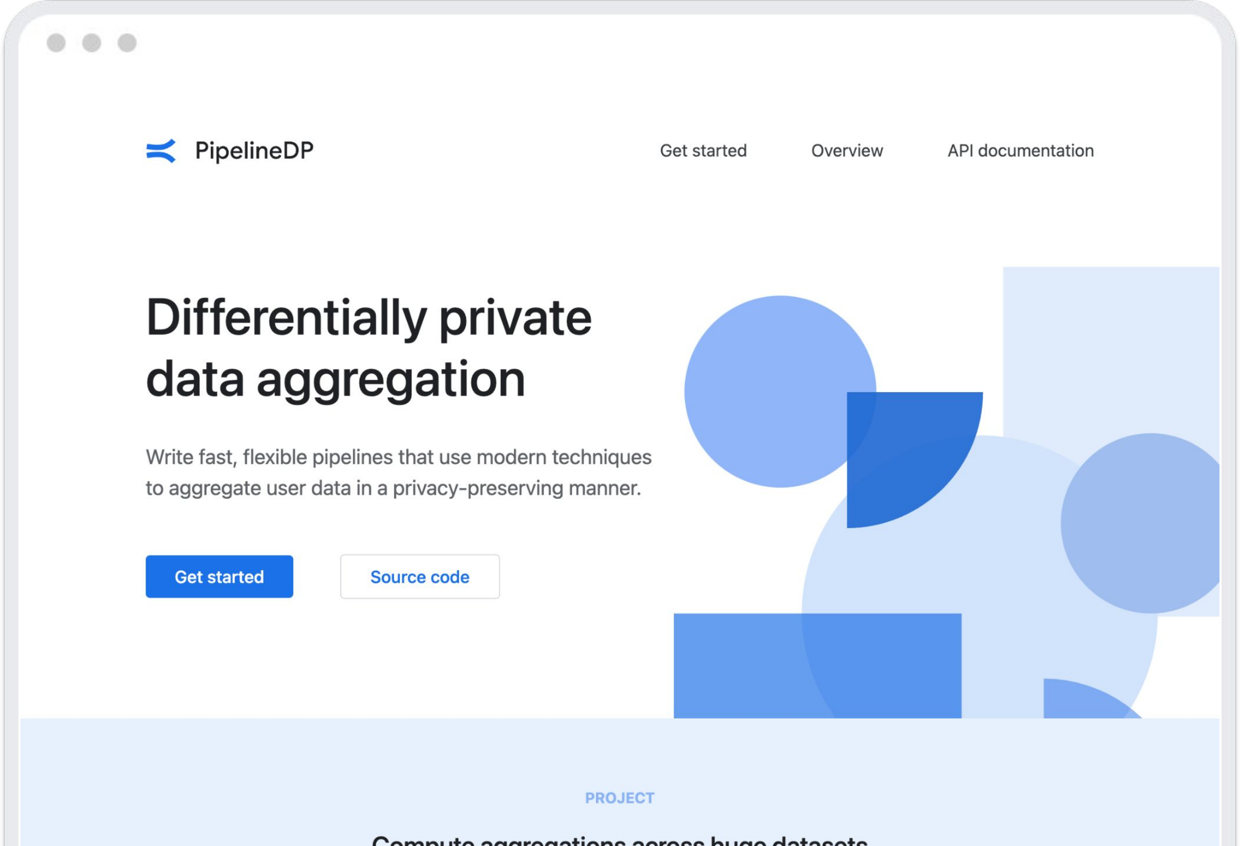
Architecture



Core DP logic:
- contribution bounding
- partition selection
- noise addition
- aggregations

map, filter, join, combine, …

pipelinedp.io



Thanks

Miguel Guevara &
Vadym Doroshenko &
Yurii Sushko

mgt@google.com, dvadym@google.com, sushko@google.com