# Open Business and Artificial Intelligence Connectivity (OBAIC)

## Overview

- Open Business and Artificial Intelligence Connectivity (OBAIC) borrows the concept from Open Database Connectivity (ODBC), which is an interface that makes it possible for applications to access data from a variety of database management systems (DBMSs). The aim of OBAIC is to define an interface allowing BI tools to access machine learning models and to run inferencing against those models on a variety of different AI platforms from a variety of AI platforms - “AI ODBC for BI”
- Through OBAIC, BI vendors can connect to any AI platform freely without concerning themselves with the underlying implementation, or how the AI platform trains the model or infers results. It's just like what we use for databases via ODBC - the caller doesn't need to concern about how the database stores the data or execute queries.
- The committee has decided this standard will only define the REST APIs protocol of how AI and BI communicate. The design or the actual implementation of OBAIC, such as whether this should be Server vs Server-less VS Docker, will be left up to the implementing vendors. If this protocol grows to another open-sourced project, that team may provide such implementation guidance/example(s).
- There are 3 key aspects designed into this standard:
    - BI - What specific call do I need this standard to provide so that I can better leverage the AI/ML platform counterpart?
    - AI - What should be the common denominator be for an AI platform that can provide support for this standard?
    - Data - Shall data be moved around in the communication between AI and BI (passed by value) or will the data remain in it's source location (passed by reference)?

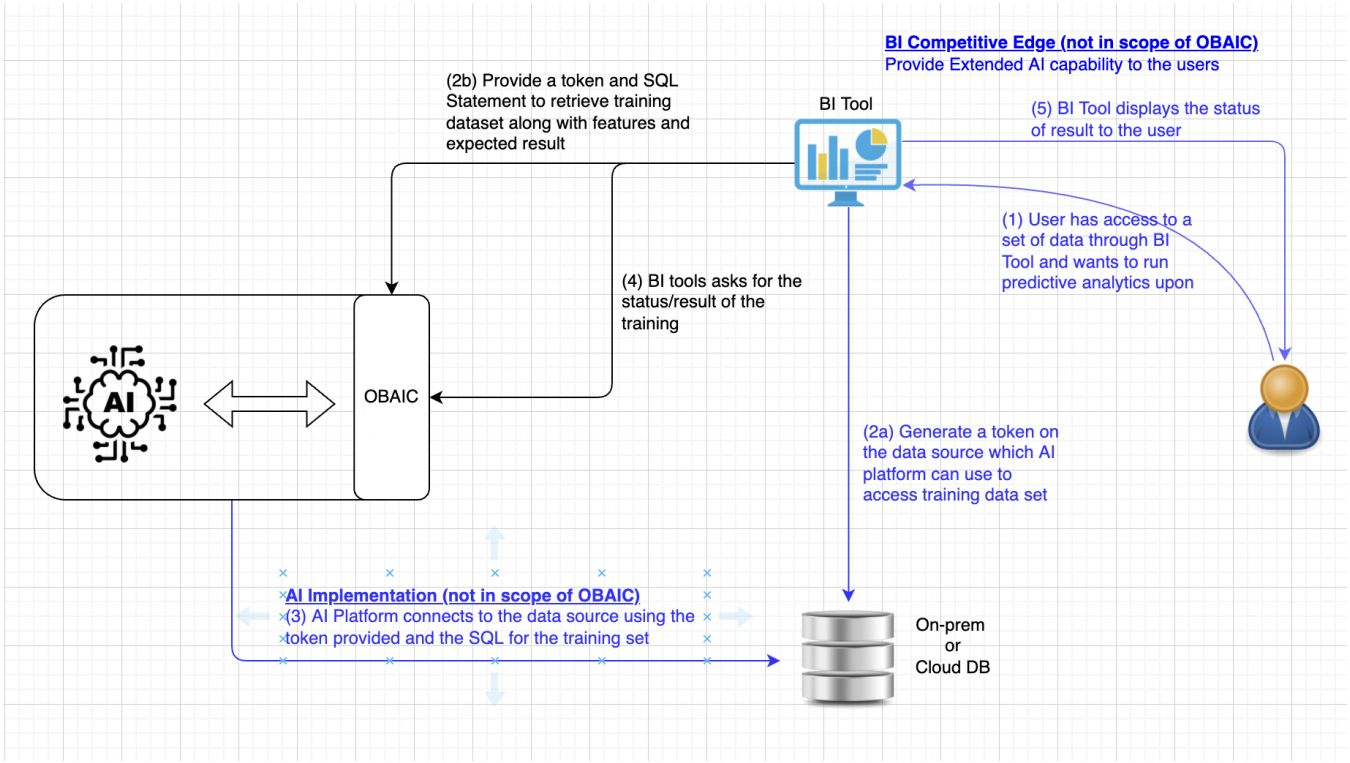
***All of the REST APIs call presented below use bearer tokens for authorization. The {prefix} of each API is configurable in the hosted servers. This protocol is inspired by Delta Sharing.***

## SwaggerHub API (Will port this document to SwaggerHub **AFTER** we confirm the design)

https://app.swaggerhub.com/apis-docs/cupidchan/OBAIC/1.0.0-oas3

## Protocol - Training

* Blue text below, either in the diagram or in the description, means it's out of scope of OBAIC and it's up to the BI tool, AI Platform or Data source vendor to implement. OBAIC is the connecting tissue to coordinate the communications among them to extend the capability of these 3 major components



1. An End User is analyzing data using their BI Tool and determines that predictive analytics for the data would be valuable and they wish to train a model with the data for that purpose.  This step is the traditional step when a user interacts with BI.
2. (a) Obtain a token with permission associated to the user making the request. This token is going to pass to AI allowing the access to the training data with a SQL statement running against the datastore. (b) BI tool, on behalf of the user, requests AI platform through OBAIC, to train/prepare a model that accepts features of a certain type (numeric, categorical, text, etc.)

    Model configuration is based on configs from the open-source Ludwig project. At a minimum, we should be able to define inputs and outputs in a fairly standard way. Other model configuration parameters are subsumed by the options field.

    The data stanza provides a bearer token allowing the ML provider to access the required data table(s) for training. The provided SQL query indicates how the training data should be extracted from the source.

    Don't be confused with the Bearer token which is used to authenticate with OBAIC, and the dbToken which is created in 2(a) and AI platform will use that to access the data source for training

| HTTP Request | Value |
|---|---|
| Method | `POST` |
| Header | `Authorization: Bearer {token}` |
| URL | `{prefix}/models/` |
| Query Parameters | `{`<br>`  "dbToken": "D41C4A382C27A4B5DF824E2D4F148";`<br>`  "inputs":[`<br>`  {`<br>`  "name":"customerAge",`<br>`  "type":"numeric"`<br>`  },`<br>`  {`<br>`  "name":"activeInLastMonth",`<br>`  "type":"binary"`<br>`  }`<br>`  ],`<br>`  "outputs":[`<br>`  {`<br>`  "name":"canceledMembership",`<br>`  "type":"binary"`<br>`  }`<br>`  ],`<br>`  "modelOptions": {`<br>`  “providerSpecificOption”: “value”`<br>`  },`<br>`  "data":{`<br>`  "sourceType":"snowflake",`<br>`  "endpoint":"some/endpoint",`<br>`  "bearerToken":"...",`<br>`  "query":"SELECT foo FROM bar WHERE baz"`<br>`  }`<br>`}` |

    If we go beyond just REST API, SQL-like is an alternative as the syntax is also well-known

    Use BigQuery ML model creation as an example and generalizing

```
CREATE MODEL (
  customerAge WITH ENCODING (
    type=numeric
  ),
  activeInLastMonth WITH ENCODING (
    type=binary
  ),
  canceledMembership WITH DECODING (
    type=binary
  )
)
FROM myData (
  sourceType=snowflake,
  endpoint="some/endpoint",
  bearerToken=<...>,
)

AS (SELECT foo FROM BAR)
WITH OPTIONS ();
```

| HTTP Response | Value |
|---|---|
| Header | `Content-Type: application/json; charset=utf-8` |
| Body | `{ "modelID": "d677b054-2cd4-4711-959b-971af0081a73" }`<br>• `modelID` is generated and returned to the caller if training is started successfully. This will be used to check the status of the training, or for future Inference (see Inference section below) |

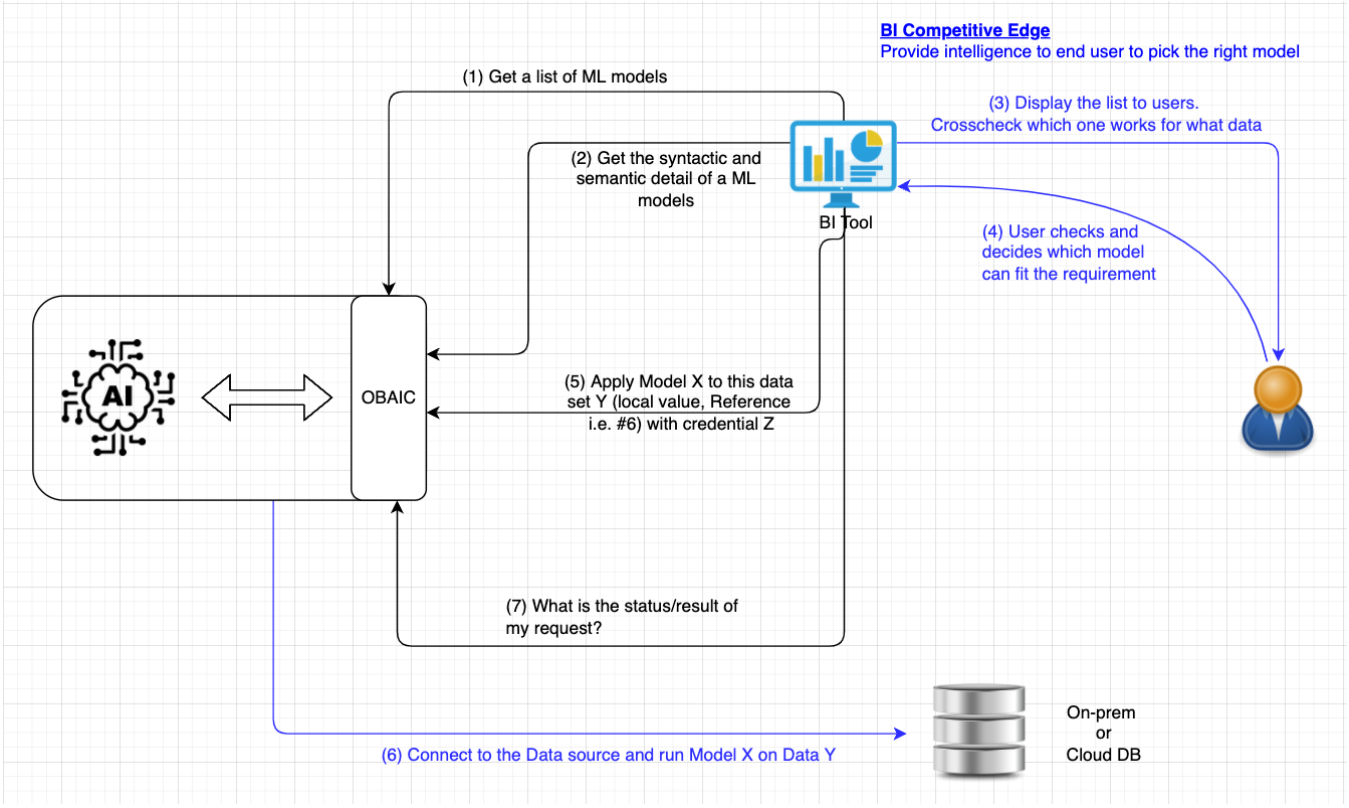
3. AI Platform provides the implementation to fulfill the request by connecting to the datasource with the provided token and the set of training data specified in SQL. This step is up to how the AI platform interacts with the data source to performance the training.
4. BI tool polls for the status or retrieve the training result. If the training is still in progress, the status will be returned. When training is completed, results and performance of the model will be returned.

| HTTP Request | Value |
|---|---|
| Method | `GET` |
| Header | `Authorization: Bearer {token}` |
| URL | `{prefix}/modelStatus?modelID=` |
| Query Parameters | **modelID** (type: String): The modelID returned from previous OBAIC call either from training or list of Models. |

| HTTP Response | Value |
|---|---|
| Header | `Content-Type: application/json; charset=utf-8` |
| Body | `{ "modelID": "d677b054-2cd4-4711-959b-971af0081a73", "status": "training", "progress": "80", }`<br>• `modelID` is same ID provided in the request<br>• `status` can be training \| inferencing \| ready<br>• `progress` is the estimated progress of the current status |

5. BI tool presents the result to the user in their own way, which is the "secret sauce" and unique to each other.

# Protocol - Inference




1. When a BI user wants to extend its capability to AI, it reaches out to AI platform and requests a list of available models of which the credential of the provided token is authorized to see

| HTTP Request | Value |
|---|---|
| Method | `GET` |
| Header | `Authorization: Bearer {token}` |
| URL | `{prefix}/models` |
| Query Parameters | **maxResults** (type: Int32, optional): The maximum number of results per page that should be returned. If the number of available results is larger than `maxResult`, the response will provide a `nextPageToken` that can be used to get the next page of results in subsequent list requests. The server may return fewer than `maxResults` items even if there are more available. The client should check `nextPageToken` in the response to determine if there are more available. Must be non-negative. 0 will return no results but ne `xtPageToken` may be populated.<br><br>**pageToken** (type: String, optional): Specifies a page token to use. Set `pageToken` to the `nextPageToken` returned by a previous list request to get the next page of results. `nextPageToken` will not be returned in a response if there are no more results available. |

| HTTP Response | Value |
|---|---|
| Header | `Content-Type: application/json; charset=utf-8` |
| Body | <pre>{<br>  "items": [<br>  {<br>  "name": "string",<br>  "id": "string"<br>  }<br>  ],<br>  "nextPageToken": "string"<br>}</pre><br>• `items` will be an empty array when no results are found.<br>• `id` field is the key to retrieved the model in the subsequent calls. Its value must be unique across the AI server and immutable through the model's lifecycle.<br>• `nextPageToken` will be missing when there are no additional results |

Example:

**GET {prefix}/models?maxResults=5**


```
{
   "models": [
      {
         "name": "Model 1",
         "id": "6d4b571a-80ca-41ef-bc67-b158f4352ad8"
      },
      {
         "name": "Model 2",
         "id": "70d9ab9d-9a64-49a8-be4d-d3a678b4ab16"
      },
      {
         "name": "Model 3",
         "id": "99914a97-5d2e-4b9f-b81a-1d43c9409162"
      },
      {
         "name": "Model 4",
         "id": "8295bfda-7901-43e8-9d31-81fd1c3210ee"
      },
      {
         "name": "Model 5",
         "id": "0693c224-3a3f-4fe7-bbbe-c70f93d15f12"
      }
   ],
   "nextPageToken": "3xXc4ZAsqZQwgejt"
}
```


2. After the list of models is returned, the BI user can selectively retrieve the detail of the model(s). This step can also be called right after the newly trained model is completed as described in the previous section since modelID is returned as a result of the training request.

| HTTP Request | Value |
|---|---|
| Method | `GET` |
| Header | `Authorization: Bearer {token}` |
| URL | `{prefix}/model/{modelID}` |
| URL Parameters | **{modelID}**: The case-insensitive ID of the model returned in in List Models for Step (1) |

| HTTP Response | Value |
|---|---|
| Header | `Content-Type: application/json; charset=utf-8` |
| Body | `{ "id": "string", "name": "string", "revision": "int", "format": { "name": "string", "version": "string" }, "algorithm": "string", // Artificial neural network \| Decision trees \| Support-vector machines \| Regression analysis \| Bayesian networks \| Genetic algorithms \| Proprietary "tags": ["string"], "dependency": "string", "creator": "string", "description": "string", "input": { "fields": [ { "name": "string", "opType": "string", "dataType": "string", "taxonomy": "string", "example": "string", "allowMissing": "boolean",` |

```
        "description": "string"
      }, ...
    ],
    "$ref": "string"
  },
  "output": {
    "fields": [
      {
        "name": "string",
        "opType": "string",
        "dataType": "string",
        "taxonomy": "string",
        "example": "string",
        "allowMissing": "boolean",
        "description": "string"
      }, ...
    ],
    "$ref": "string"
  },
  "performance": {
    "metric": "string",
    "value": "float"
  },
  "rating": "int",
  "url": "string"
}
```

- `format.name`: PMML, ONNX, or other formats to be confirmed
- `algorithm`: Artificial neural network | Decision trees | Support-vector machines | Regression analysis | Bayesian networks | Genetic algorithms | Proprietary
- `tags`: describe what this model is used for e.g. Agriculture | Banking | Computer vision | Credit-card fraud detection | Handwriting recognition | Insurance | Machine translation | Marketing | Natural language processing | Online advertising | Recommender systems | Sentiment analysis | Telecommunication | Time-series forecasting | etc.
- `opType`: categorical | ordinal | continuous
- `dataType`: string | integer | float | double | boolean | date | time | dateTime
- `$ref`: reference to external schema for the format used
- `metric`: based on model used, metric can be accuracy, precision, recall, ROC, AUC, Gini coefficient, Log loss, F1 score, MAE, MSE, etc.
- `url`: link to the real model for download

Example:

**GET {prefix}/models/6d4b571a-80ca-41ef-bc67-b158f4352ad8**

```
{
    "id": "6d4b571a-80ca-41ef-bc67-b158f4352ad8",
    "name": "Model 1",
    "revision": 3,
    "format": {
      "name": "PMML",
      "version": "4.3"
    },
    "algorithm": "Neural Network",
    "tags": [
      "Anomaly detection",
      "Banking"
    ],
    "dependency", "",
    "creator": "John Doe",
    "description": "This is a predictive model, refer to {input} and {output} for detailed format of each
field, such as value range of a field, as well as possible predictions the model will gave. You may also refer
to the example data here.",
    "input": {
      "fields": [
        {
          "name": "Account ID",
          "opType": "categorical",
          "dataType": "string",
          "taxonomy": "ID",
          "example": "account abc-001",
          "allowMissing": false,
          "description": "unique value"
```

```
        },
        {
          "name": "Account Balance",
          "opType": "continuous",
          "dataType": "double",
          "taxonomy": "currency",
          "example": "1,378,560.00",
          "allowMissing": true,
          "description": "Minimum: 0, Maximum: 999,999,999.00"
        },
      ],
      "ref": "http://dmg.org/pmml/v4-3/pmml-4-3.xsd"
    }
    "output": {
      "fields": [
        {
          "name": "Churn",
          "opType": "continuous",
          "dataType": "string",
          "taxonomy": "ID",
          "example": "0.67",
          "allowMissing": false,
          "description": "the possibility of the account stop doing business with a company over 6 months"
        }
      ],
      "ref": "http://dmg.org/pmml/v4-3/pmml-4-3.xsd"
    }
    "performance": {
      "metric": "accuracy",
      "value": 0.85
    },
    "rating": 5,
    "url": "uri://link_to_the_model"
}
```

3. The BI tools will use the information retrieved from the AI platform to display to the user, including what type of models are available and the performance. It can optionally match the data and suggest what may be the good match based on what the user has.

4. User interacts with the result BI presented and decides what can be a good model to make a prediction on certain set of data. Please note that the model can also be returned as the result of the training step described in the previous section. In the case, the user may bypass these 2 steps and go directly to see the result.

5. Once the BI user/developer decides which model to run for predictions, they will take the appropriate actions in the BI tool to prepare the data and call OBAIC and request it run that model with the data.

**Query should be in the body**

| HTTP Request | Value |
|---|---|
| Method | `POST` |
| Header | `Authorization: Bearer {token}` |
| URL | `{prefix}/models/{modelID}` |
| Query Parameters | `{`<br><br>`"dbToken": "string";`<br>`"action": "string",`<br>`"data":{`<br>`"sourceType":"string",`<br>`"endpoint":"string",`<br>`"bearerToken":"string",`<br>`"query":"string"`<br>`}`<br>`}` |
| Example | `{` |

| | |
|---|---|
| | <pre>"dbToken": "D41C4A382C27A4B5DF824E2D4F148";<br>"action": "infer",<br>"data":{<br>"sourceType":"snowflake",<br>"endpoint":"some/endpoint",<br>"bearerToken":"7CA4D3C152646DDEFB527A958C45B",<br>"query":"SELECT foo FROM bar WHERE baz"<br>}<br>}</pre> |

**Explanation**

| HTTP Response | Value |
|---|---|
| Header | `Content-Type: application/json` |
| Body | <pre>{<br>"resultStatus": "ready",<br>"result":{<br>"sourceType":"snowflake",<br>"endpoint":"some/endpoint",<br>"bearerToken":"7CA4D3C152646DDEFB527A958C45B",<br>"query":"SELECT * FROM resultTable"<br>}<br>}</pre> |

Pass by value is recommended only for small data sets

| HTTP Request | Value |
|---|---|
| Method | `GET` |
| Header | `Authorization: Bearer {token}` |
| URL | `{prefix}/models/model/{modelID}` |
| Query Parameters | <pre>{<br>"action": "string",<br>"data":{<br>"sourceType":"string",<br>"endpoint":"string",<br>"bearerToken":"string",<br>"query":"string"<br>}<br>}</pre> |
| Example | <pre>{<br>"action": "infer",<br>"data":[<br>{<br>"AccountBalance": 100,<br>"YearOpened": 1990<br>},<br>{<br>"AccountBalance": 200,<br>"YearOpened": 1995<br>},<br>]<br>}</pre> |

| HTTP Response | Value |
|---|---|
| Header | `Content-Type: application/json` |
| Body | <pre>{<br>"resultStatus": "ready",<br>"result":[<br>{<br>"AccountBalance": 100,<br>"YearOpened": 1990,</pre> |

<table>
<tr><td></td><td><pre>
      "Churn": 80
    },
    {
      "AccountBalance": 200,
      "YearOpened": 1995,
      "Churn": 40
    },
  ]
}
</pre></td></tr>
</table>

| HTTP Response | Value |
|---|---|
| Header | `Content-Type: application/json` |
| Body | <pre>{<br>  "resultStatus": "notReady",<br>  "resultProgress": 50,<br>  "resultID": "925D2129FC57FAC5"<br>}</pre> |

6. AI will connect to the underlying data source and run the prediction using the information provided by BI

7. In case the result cannot be returned immediately because of the prediction volume, BI can poll for the result.

In case of pass by reference

**GET {prefix}/results/925D2129FC57FAC5**

```
{
  "resultStatus": "ready",
  "result":{
    "sourceType":"snowflake",
    "endpoint":"some/endpoint",
    "bearerToken":"7CA4D3C152646DDEFB527A958C45B",
    "query":"SELECT * FROM resultTable"
  }
}
```

In case of pass by value:

**GET {prefix}/results/925D2129FC57FAC5**

```
{
  "resultStatus": "ready",
  "result":[
    {
      "AccountBalance": 100,
      "YearOpened": 1990,
      "Churn": 80
    },
    {
      "AccountBalance": 200,
      "YearOpened": 1995,
      "Churn": 40
    },
  ]
}
```

# Error - Apply to all API calls above

| HTTP Response | Value |
|---|---|
| Header | `Content-Type: application/json` |
| Body | <pre>{<br>"errorCode": "string",<br>"message": "string"<br>}</pre> |

| HTTP Response | Value |
| --- | --- |
| Header | `Content-Type: application/json` |
| Body | `{ "errorCode": "string", "message": "string" }` |

| HTTP Response | Value |
| --- | --- |
| Header | `Content-Type: application/json` |
| Body | `{ "errorCode": "string", "message": "string" }` |

| HTTP Response | Value |
| --- | --- |
| Header | `Content-Type: application/json` |
| Body | `{ "errorCode": "string", "message": "string" }` |

# Next Step

- Finalize Logo
- Determine what other AI model format can be supported by OBAIC besides ONNX, Neuropod and PMML per https://landscape.lfai.foundation/card-mode?category=format-interface&grouping=category)

# Future Enhancement

- Define potential value for each parameter in the API call
- Formally define JSON in http://json-schema.org/ so that future development can validate the JSON structure
- Define data pipeline to transform data before running
- Define containerized model so that prediction can run in BI instead of in AI
- Define format of nextPageToken
- Define different types of `errorCode` and `message` for each API call

# FAQ

1. Why should AI vendors care to participate in the OBAIC standard?
    - Most AI model training and execution is done by a very small set of data scientists. The OBAIC protocol extends the influence and ability for the AI vendors. They will no longer need to work with BI partners to create one off implementations and continually maintain those.
2. Why should BI vendors care to participate in the OBAIC standard?
    - Providing end users access to predictions/prescriptions is the desired goal. Writing one off drivers to support every AI flavor of the week isn't the goal. The OBAIC standard provides BI vendors the opportunity to build and support 1 driver, while then enabling all customers /prospects to bring their own AI implementation(s) to the table and not lose deals as the result of not having a customer driver in place for that customer to expand, or prospect to purchase.
3. Who owns the model and data?
    - The AI platform owns the models but share those with BI tools through OBAIC. The data is owned by the business but BI has been authorized to use it and re-share this to AI for training and inference.
4. How do you deal with Security?
    - Call will be handled by HTTPS protocol and authorized by bearer token standard

# References

- Tableau version of OBAIC https://tableau.github.io/analytics-extensions-api/docs/ae_example_tabpy.html
- Qlik version of OBAIC: https://github.com/qlik-oss/server-side-extension
- Delta Sharing: https://github.com/delta-io/delta-sharing/blob/main/PROTOCOL.md#delta-sharing-protocol

# Authors


| Name | Affiliation |
| --- | --- |
| Cupid Chan | Pistevo Decision |
| Xiangxiang Meng | Upstart |
| Deepak Karuppiah | MicroStrategy |
| Dalton Ruer | Qlik |
| Sachin Sinha | Microsoft |
| Yi Shao | IBM |
| Stu Sztukowski | SAS |
| Jeffrey Tang | Predibaes |
| Lingyan Yin | Salesforce |