MDC Workshop (II)

Data, Algorithm & Computation, Creativity

Janet Huang

2023.03.13

Summary of Feedback

Good

The course was very hands-on explorative, and that got me excited about learning more.

Interesting introduction of multiple tools. Useful suggestions

Learn about how to use AI tools, and play with it.

I liked that we got support in the prototyping and how we could apply this ourselves in the project.

Relevant coding available for own projects (and later referencing!)

Need to be improved

1. More time for your coding practice

More time for the coding session

The hands-on part in the end could have been longer so we could try out ourselves a bit more!

Maybe it's a bit too long, if there are more people helping out when you're stuck it might be faster

2. We have Eden for technical support

context; what kind of applications are online generators that we can easily use, and what kind of applications require coding by ourselves and are rather complicated. And more elaborate on the coding

Step by step more slowly in coding part

Guide us slower through the code, also many terms and actions were not clear or new so I had no clue what I was actually doing

3. Step-by-step code explanation, context, and why

MDC Workshops

MDC Workshop (I): Basic (3 hrs)

1. Introduction of thingCV (40 mins)
- Introduction (10 mins)
- Exercise 1: play with thingCV (20 mins)
- Sharing (5-7 mins)

2. Collaborate with thingCV and generative AI (30 mins)
- Introduction (5 mins)
- Exercise 2: play with generative AI (20 mins)
- Sharing (5 mins)

3. Build your thingCV from scratch (I) (90 mins)
- Foundation: ml5.js, Data Foundry, observable
- Exercise 3: object hunter
- Exercise 4: data visualizer, ~~data exploration~~

MDC Workshop (I): Advanced (2 hrs)

1. Build your thingCV from scratch (II): data, computation, algorithm **(30 mins)**
- Foundation: integration of object hunter, network builder, and data visualizer (Data Foundry, starboard, d3.js)
- Exercise 1: try out starboard,

2. Build your LLMs model on DF **(1.5 hrs)**
- Foundation: OpenAI api in Data Foundry
- Exercise 2: create an interface for interacting with OpenAI api

MDC Workshops

MDC Workshop (I): Basic (3 hrs)

1. Introduction of thingCV (40 mins)
- Introduction (10 mins)
- Exercise 1: play with thingCV (20 mins)
- Sharing (5-7 mins)

2. Collaborate with thingCV and generative AI (30 mins)
- Introduction (5 mins)
- Exercise 2: play with generative AI (20 mins)
- Sharing (5 mins)

3. Build your thingCV from scratch (I) (90 mins)
- Foundation: ml5.js, Data Foundry, observable
- Exercise 3: object hunter
- Exercise 4: data visualizer, ~~data exploration~~

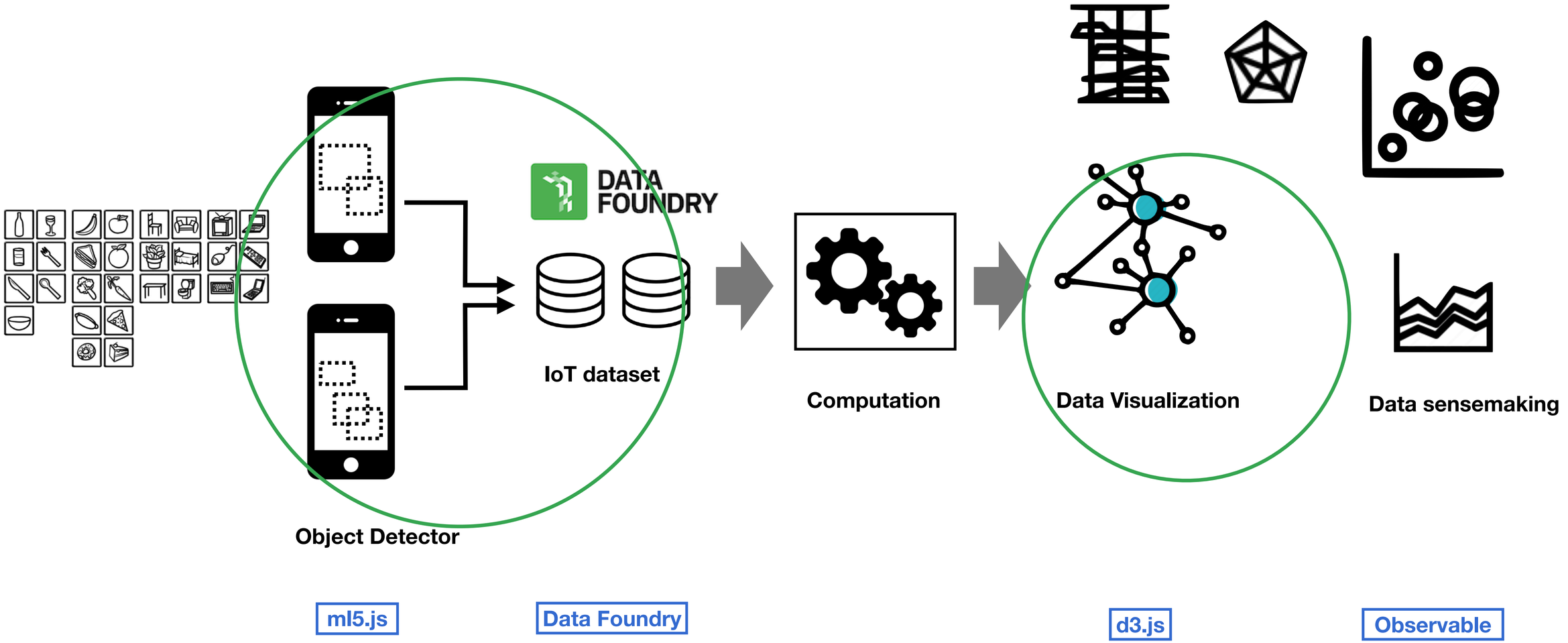
MDC Workshop (I): Advanced (2 hrs)

1. Build your thingCV from scratch (II): data, computation, algorithm **(30 mins)**
- Foundation: integration of object hunter, network builder, and data visualizer (Data Foundry, starboard, d3.js)
- Exercise 1: try out starboard,

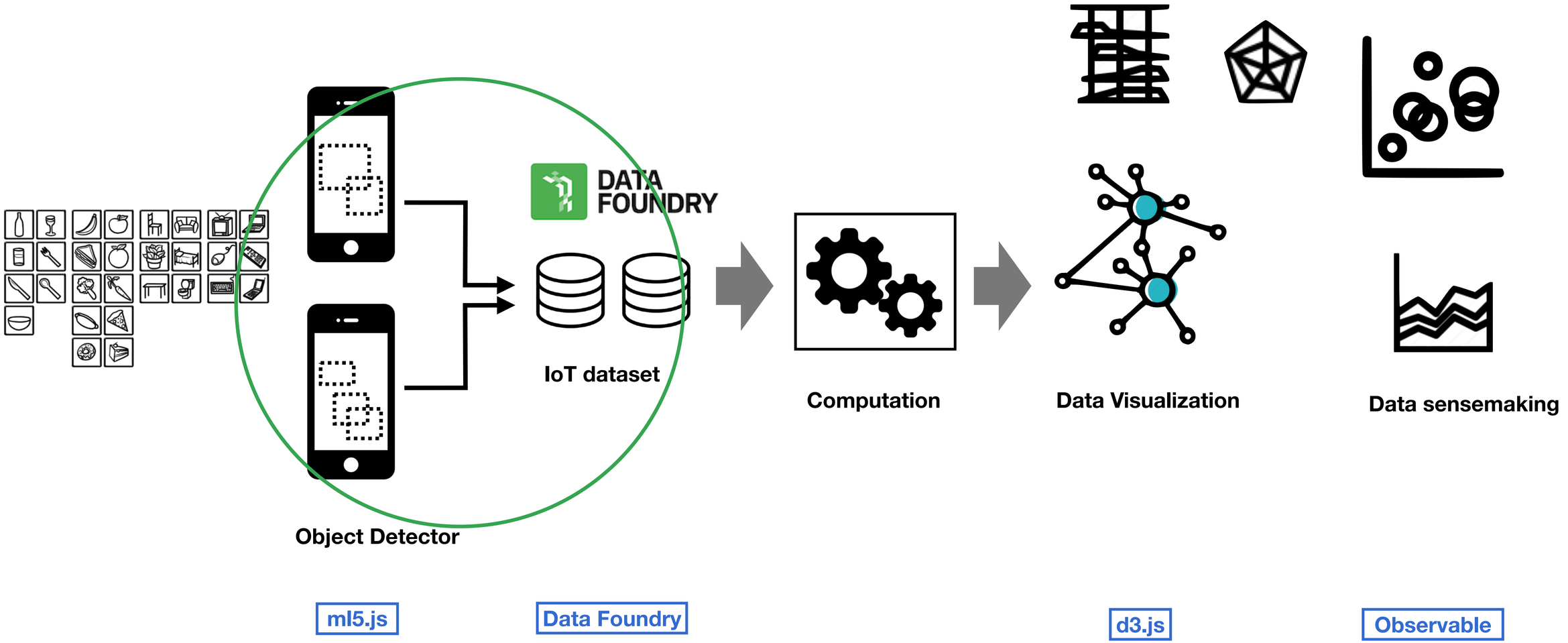
2. Build your LLMs model **(1.5 hrs)**
- Foundation: OpenAI api in Data Foundry
- Exercise 2: create an interface for interacting with OpenAI api

Recap: Object detector + Data visualizer

System Workflow



System Workflow



editor.p5js.org

gmail Research agent wiki Good Sites Class Tools Dict Design tool oTranscribe Time Zone academics - Dropbox TU/e 圖庫&Icon

p5.js Web Editor | MDC_Workshop_1_ObjectDetector_Webcam

p5* File Edit Sketch Help English Hello, janetyc!

Auto-refresh MDC_Workshop_1_ObjectDetector_Webcam by janetyc

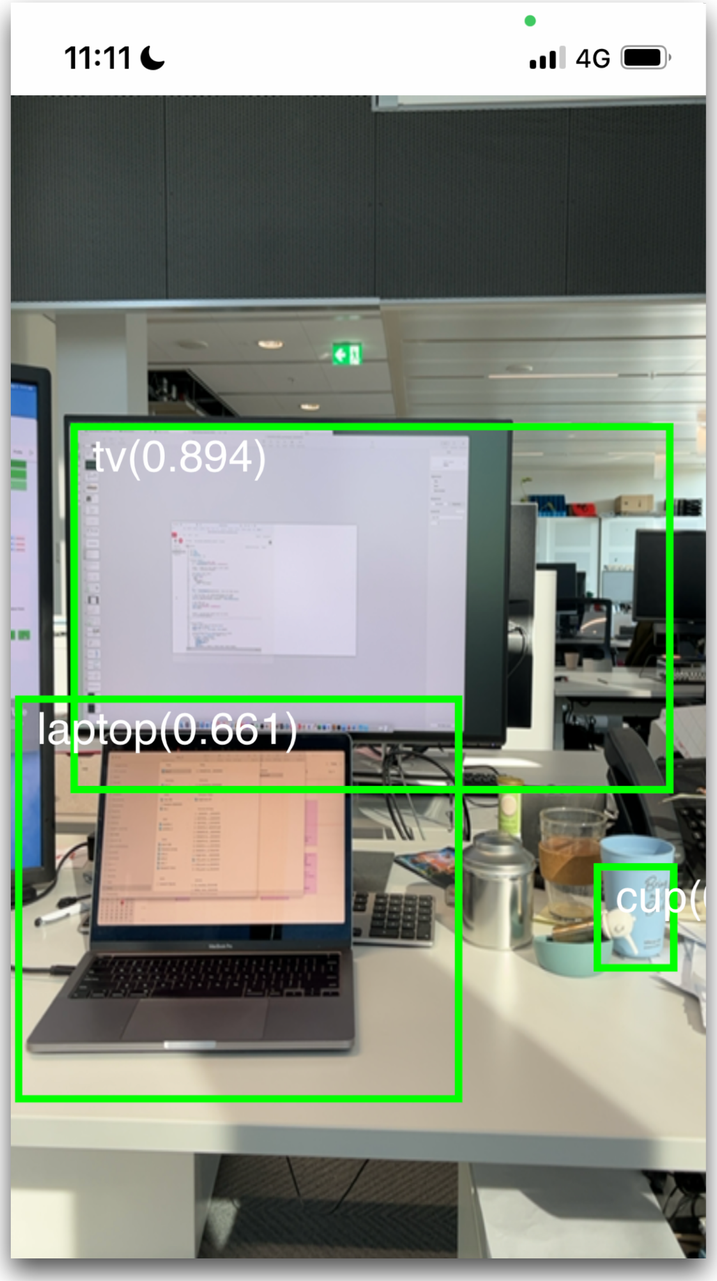
Sketch Files

- index.html
- sketch.js
- style.css

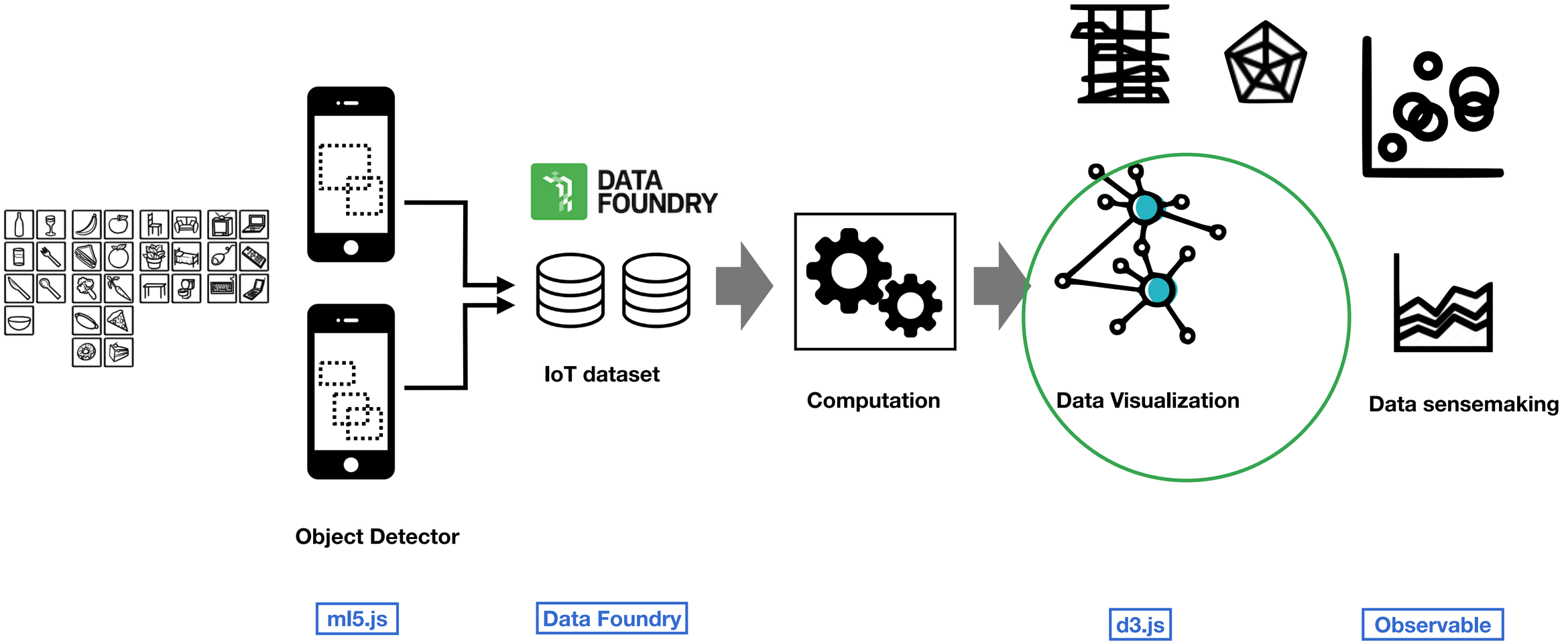
sketch.js — Saved: about 18 hours ago — Preview

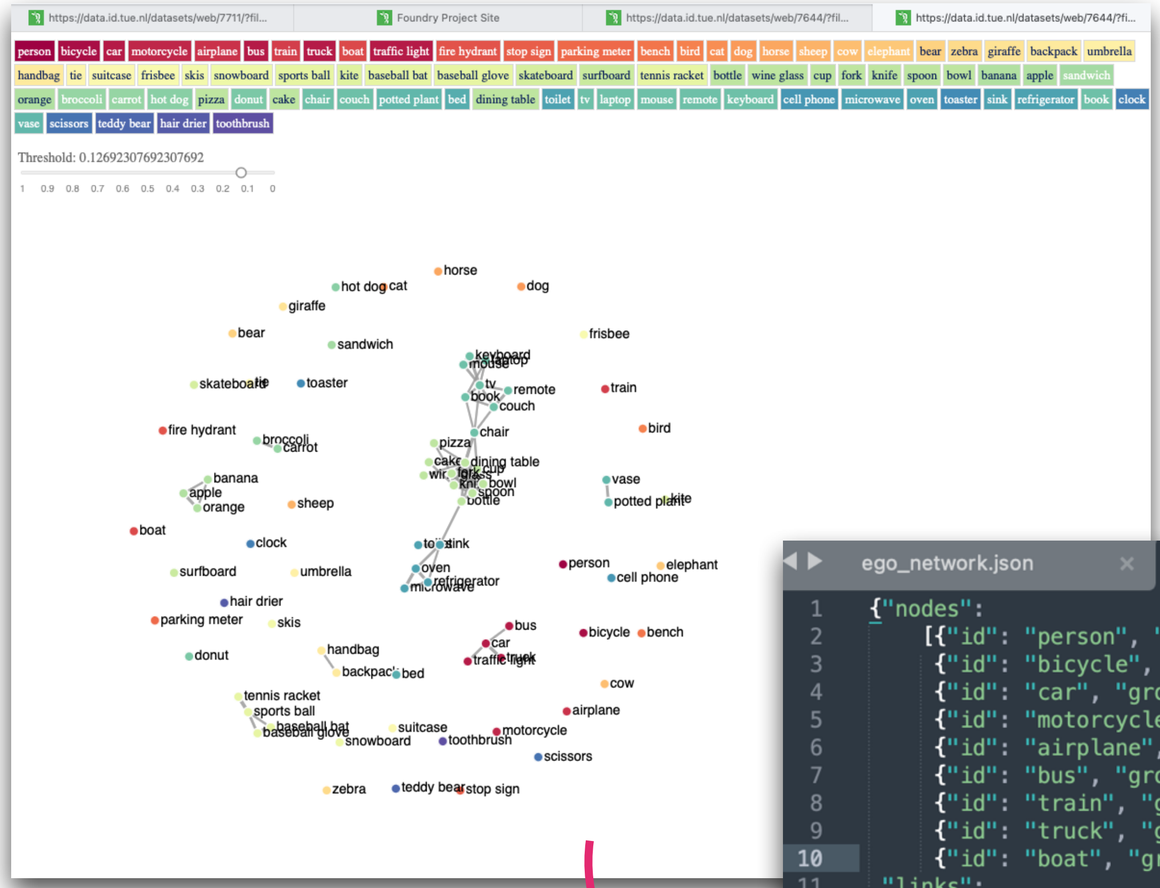
```
let cnv;
let video;
let detector;
let detections = [];

function setup() {
 // cnv = createCanvas(500, 500);
  cnv = createCanvas(windowWidth, windowHeight);

  //Step 1: change the front camera to back camera
  //video = createCapture(VIDEO);

  //use phone's back camera
  var constraints = {
    audio: false,
    video: {
      facingMode: {
        exact: "environment"
      }
    }
  };
  video = createCapture(constraints); //save the video results

  // The line below + the videoLoadedCallback were added
  // after the video was shot to fix compability issues.
  video.elt.addEventListener('loadeddata', videoLoadedCallback);

  // set video size
  //video.size(500, 500);
  video.size(windowWidth, windowHeight);
  video.hide();


  //Step 3: log data when people touch the screen
  //cnv.touchEnded(logData);

}

function draw() {
  //Step 2: draw video and detected objects
  //draw video on canvas
  image(video, 0, 0, video.width, video.height);

  //draw bounding boxes for detected objects on canvas
  for (let i = 0; i < detections.length; i++) {
    let object = detections[i];
    if(object.confidence > 0.6) {
      stroke(0, 255, 0);
      strokeWeight(4);
      noFill();
      rect(object.x, object.y, object.width, object.height);
```

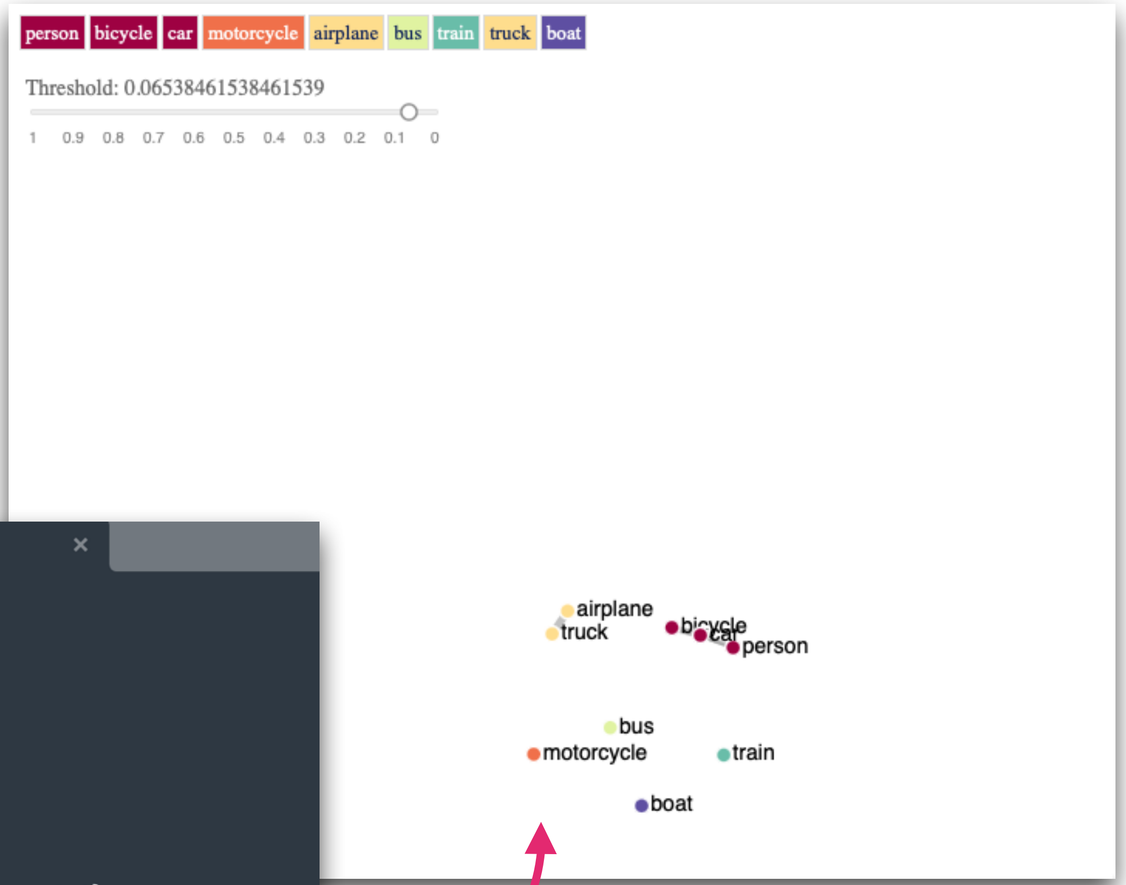
Console Clear

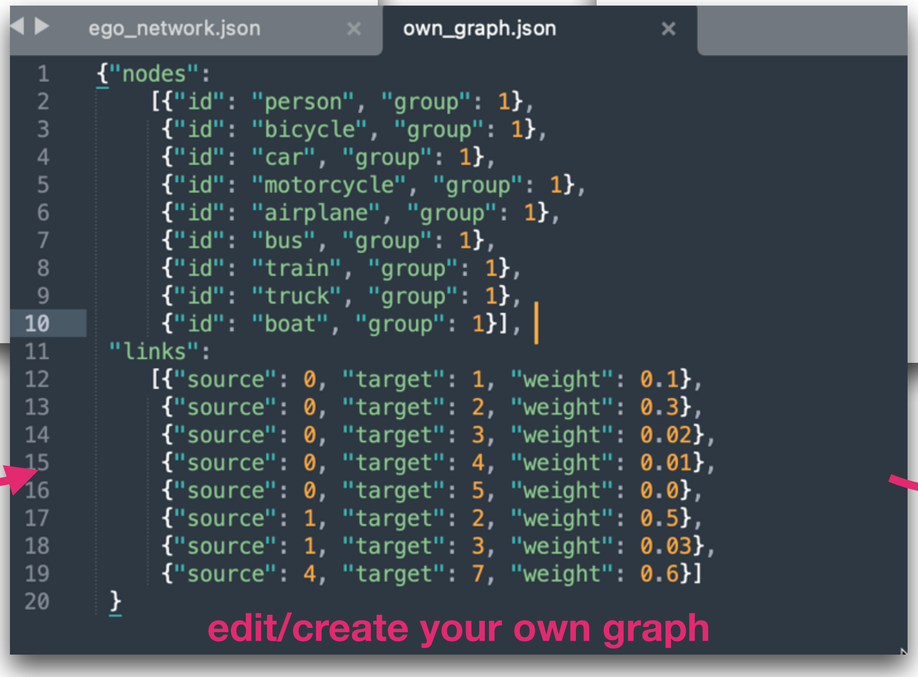


System Workflow



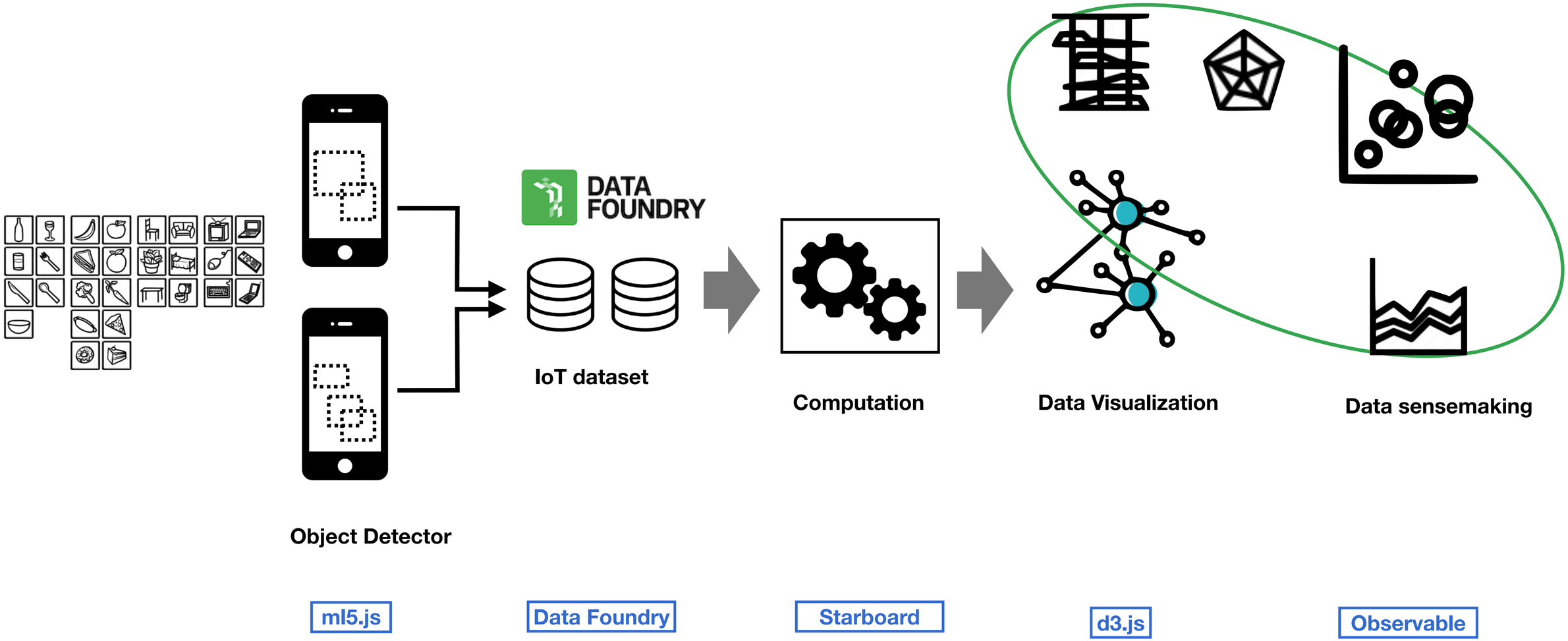






own_graph.json

System Workflow



Data Visualization

observablehq.com

gmail Research agent wiki Good Sites Class Tools Dict Design tool oTranscribe Time Zone academics - Dropbox TU/e 圖庫&Icon

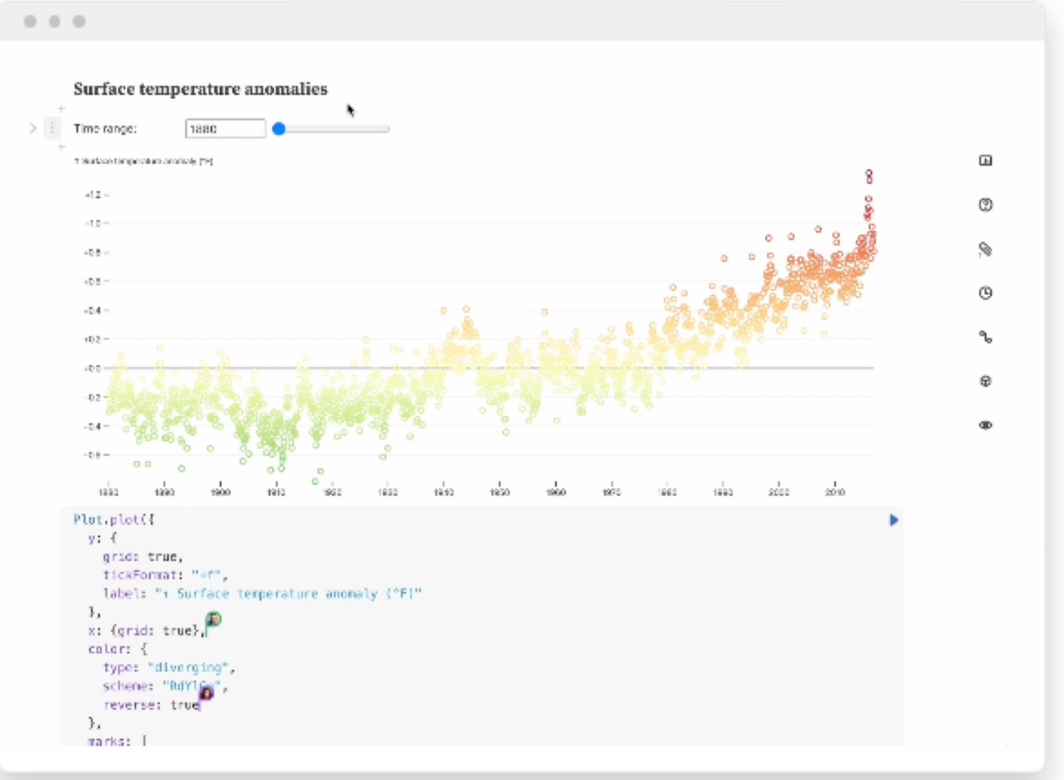
Collaborative data platform and canvas | Observable

Product Pricing Solutions Explore Learn Community Search Sign in Sign up

Explore, analyze, and explain data. **As a team.**

Uncover new insights, answer more questions, and make better decisions.

Sign up for free



https://observablehq.com/

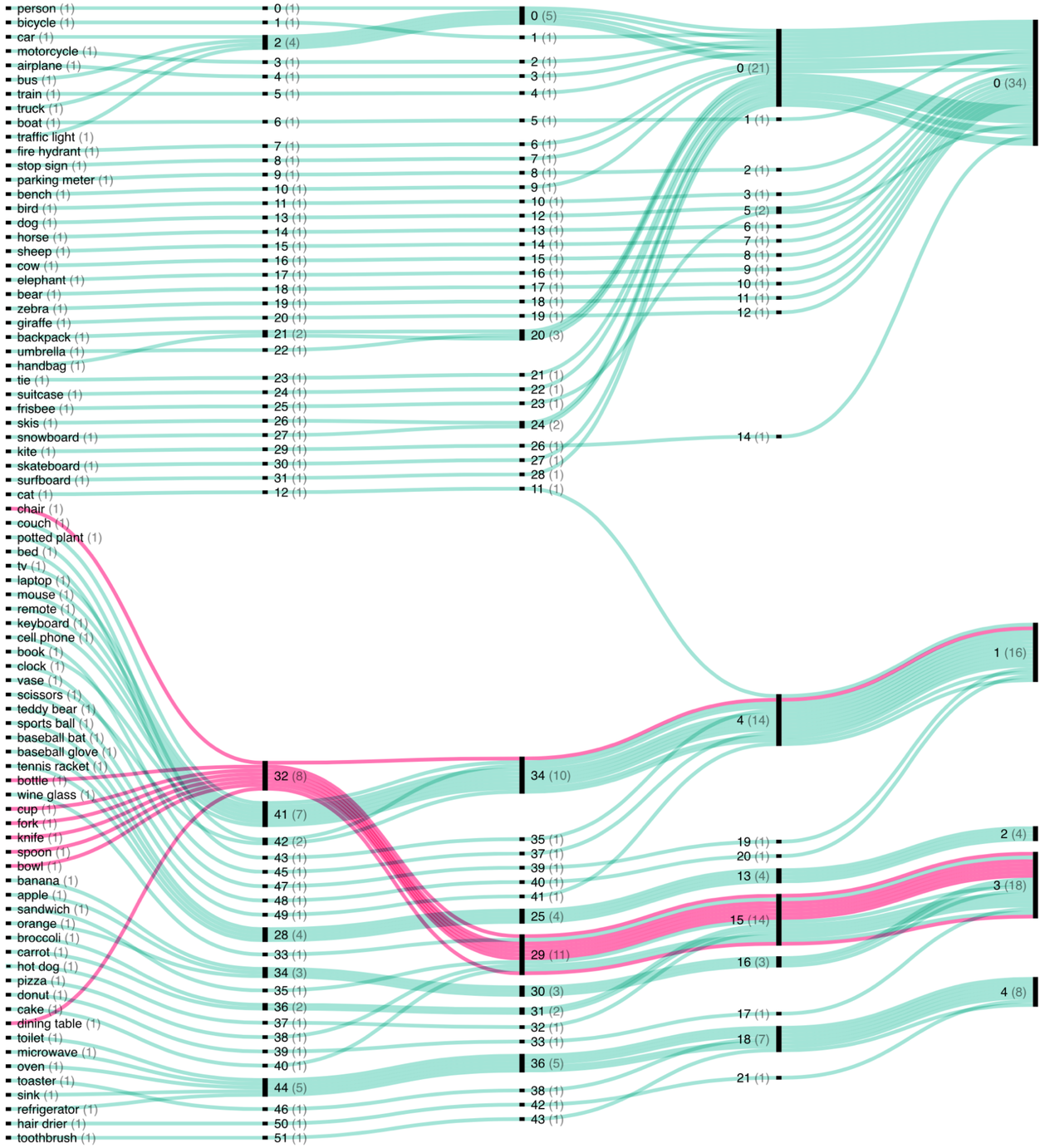
Arc Diagram

This diagram places nodes in a horizontal or vertical line, with circular arcs for links. Unlike other network visualizations such as a force layout, the appearance (and usefulness) of an arc diagram is highly dependent on the order of nodes. Hover over a node below to inspect its connections.

https://observablehq.com/@janetyc/arc-diagram

Visualization: Parallel sets

Parallel sets are like parallel coordinates, but for categorical dimensions. The thickness of each curved line represents a quantity that is repeatedly subdivided by category. This example looks at community flow of Everyday objects in thingCV



https://observablehq.com/@janetyc/community-flow-of-everyday-objects

Visualization: Radial Tidy Tree

D3's tree layout implements the Reingold–Tilford "tidy" algorithm for constructing hierarchical node-link diagrams, improved to run in linear time by Buchheim *et al.* Tidy trees are typically more compact than cluster dendrograms, which place all leaves at the same level. See also the Cartesian variant.

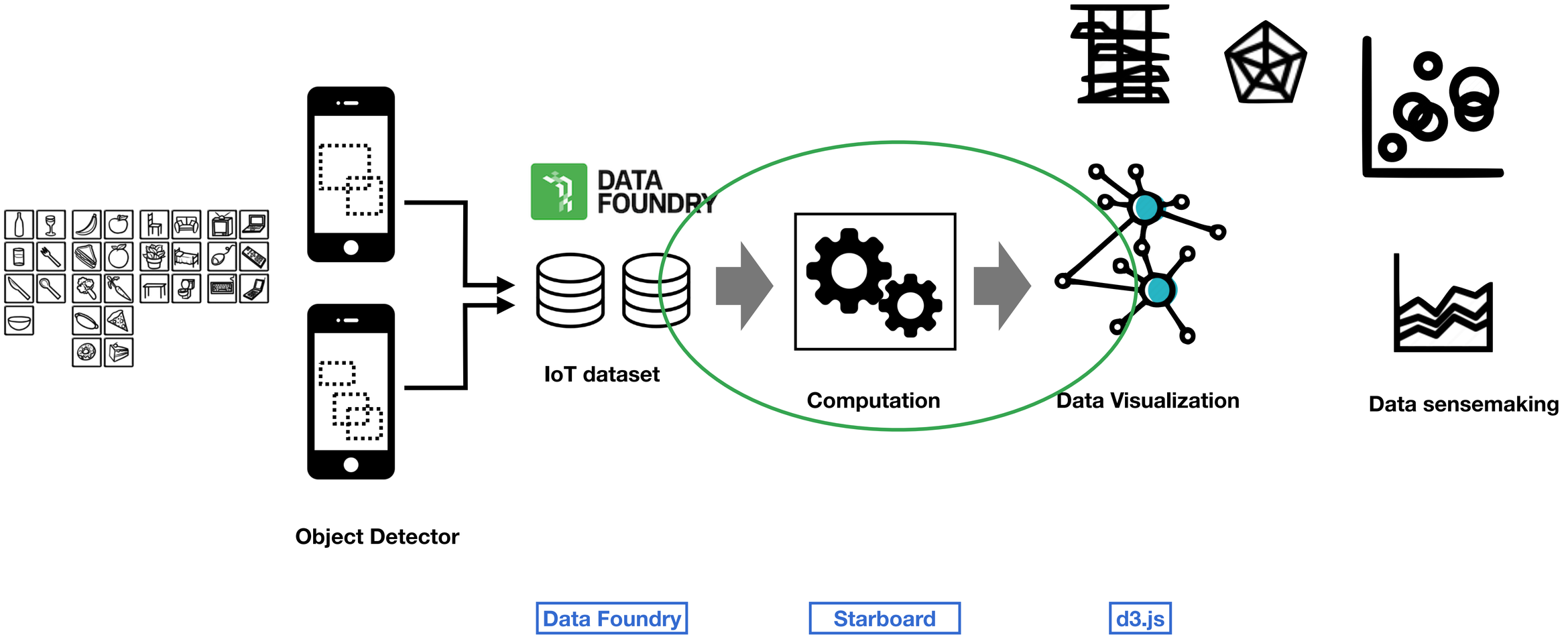
dataset: https://visualgenome.org/

visualization: https://observablehq.com/@janetyc/radial-tidy-tree-for-object-story-visualization

Session I: Integration of “object detector”, “network builder”, and “data visualizer”

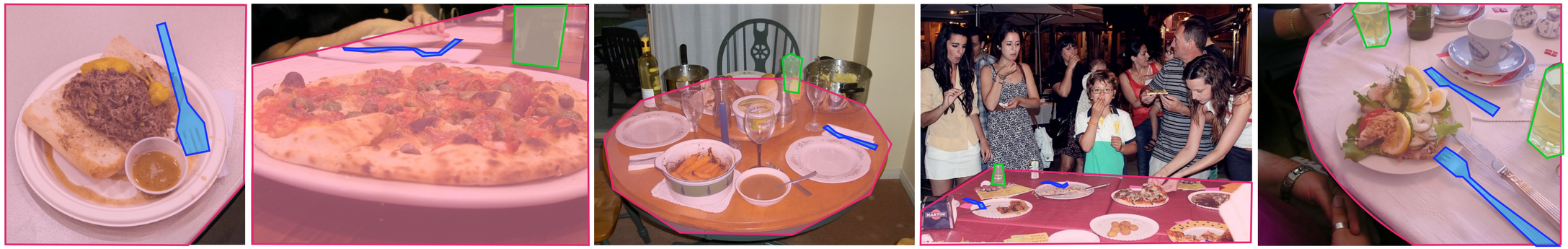
(1) Read data from Data foundry
(2) Build object network based on collected data through similarity calculation (starboard on DF)
(3) Integrate all components (hosting a web-based application on DF)

System Workflow



Construct a Social Network of Object Based on Object Co-occurrence

cup
fork
dining table

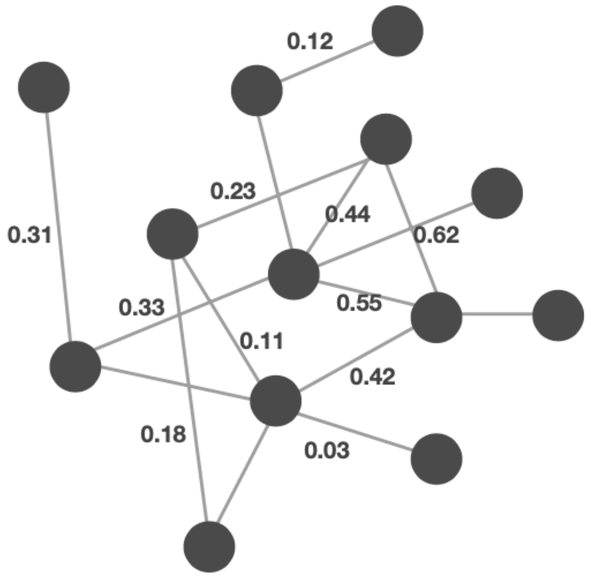


$$Relation(Cup, Fork) = \frac{|\, Collection_{Cup} \cap Collection_{Fork} \,|}{|\, Collection_{Cup} \cup Collection_{Fork} \,|}$$

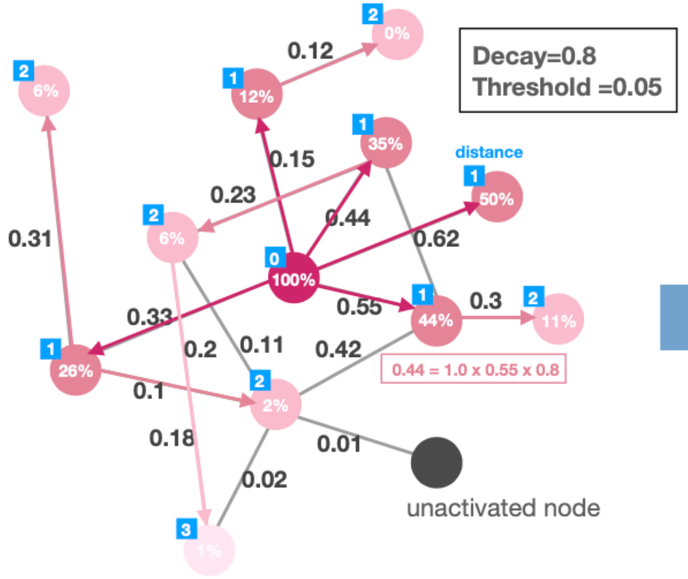
4/5 = 0.80

(Jaccard similarity coefficient)

Construct Social-Centric and Ego-Centric Network

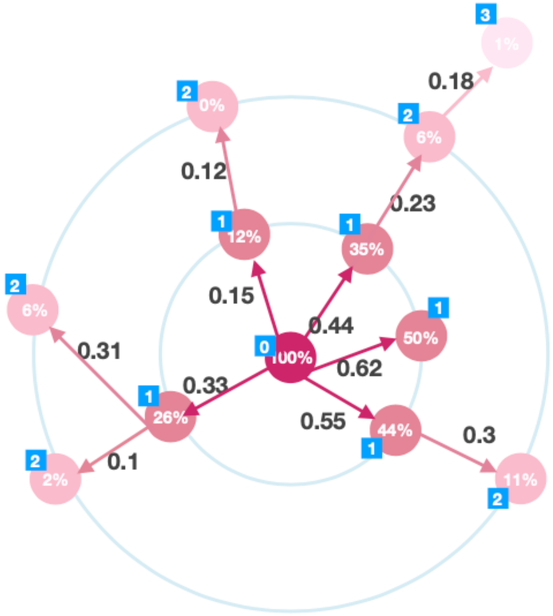


Social-Centric Network (undirected)



Spreading Activation

$$Energy(o_j) = Energy(o_i) * Relation(o_i, o_j) * Decay$$



Ego-Centric Network (directed)

Starboard

The shareable in-browser notebook

Support Markdown, Latex, HTML, CSS, Javascript, and Python

View source

EXAMPLE NOTEBOOKS

- Introduction to Starboard
- Data Visualization in JS
- Python: Pandas and Matplotlib
- Procedural Art

Starboard is an Open Startup.

Starboard is in beta, you can read more about Starboard here.

MARKDOWN

H1 # Introducing Starboard Notebook

Starboard brings cell-by-cell notebooks to the browser, no code is running on the backend here!
It's probably the quickest way to visualize some data with interactivity, do some prototyping, or build a rudimentary dashboard.

H4 **Some features**

- Mix Markdown, LaTeX, HTML, CSS, Javascript, and Python.
- The file format is a plaintext file, which plays nice with version control systems like git.
- Runs entirely in your browser, everything is static: no server, no setup and no build step.
- You can embed a fully functional notebook on your website.

Let's see it in action!

> *Tip: Press the ▶ Play button on the left to run a cell's code.*

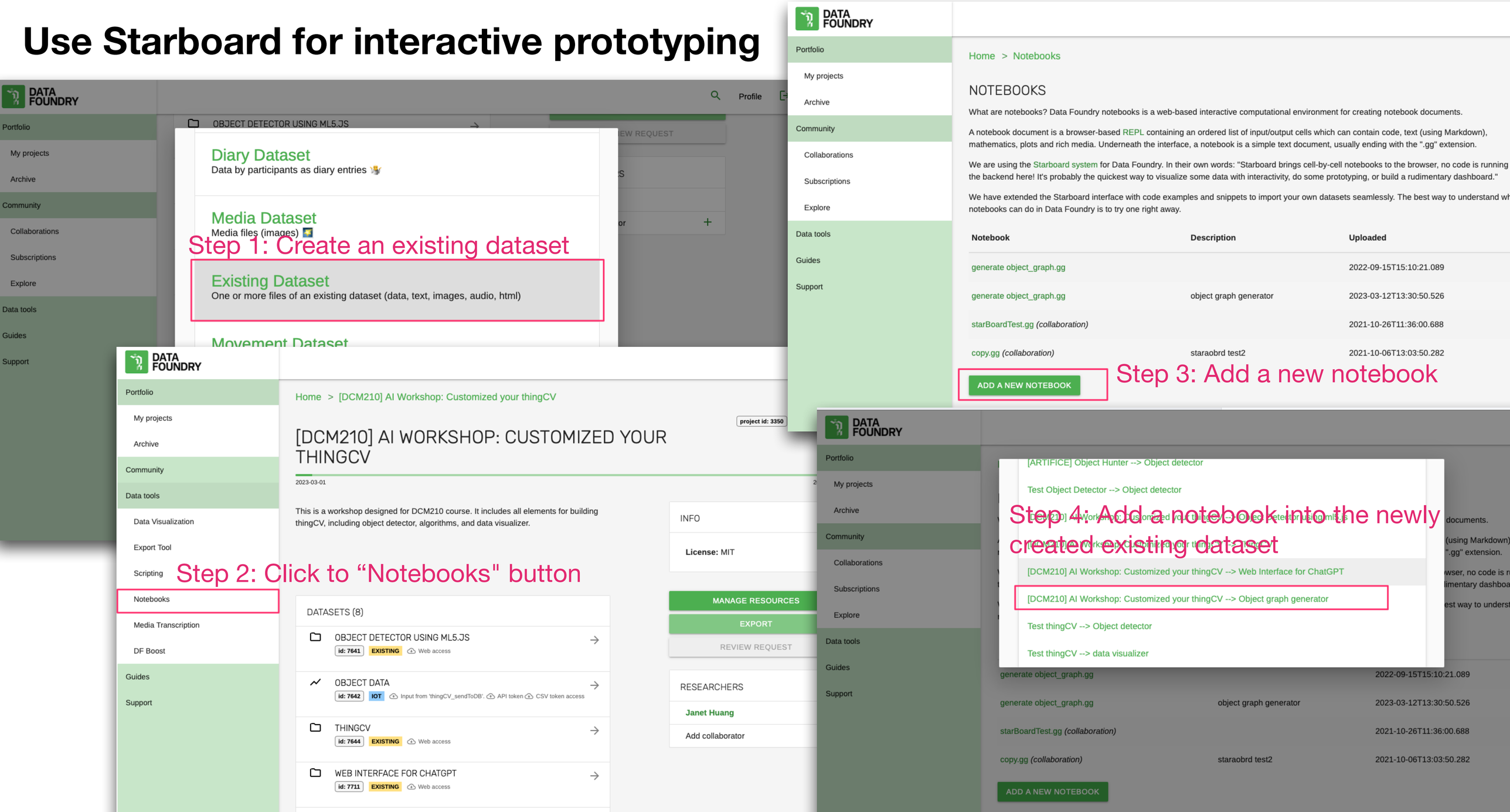
</>

JAVASCRIPT

```
1  // You write vanilla Javascript
2  const greeting = "Hello world!";
3
4  // The last statement in a cell will be displayed if it is not undefined.
5  greeting
```

https://starboard.gg/

Use Starboard for interactive prototyping



DATA FOUNDRY

Profile

- Portfolio
  - My projects
  - Archive
- Community
  - Collaborations
  - Subscriptions
  - Explore
- Data tools
- Guides
- Support

WEB INTERFACE FOR CHATGPT
id: 7711 EXISTING Web access

CHATGPT SCRIPT
id: 7712 SCRIPT Script is installed on 'callChatAPI'.

_TESTRANDOMDATA
id: 7713 SCRIPT Script is inactive.

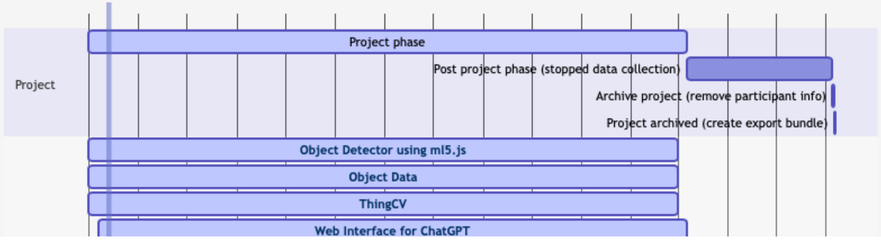
CONTINUE-CHAT
id: 7729 SCRIPT Script is inactive.

OBJECT GRAPH GENERATOR
id: 7730 EXISTING

ADD DATASET OR SCRIPT
choose a dataset or script to add

Manage participants, wearables and devices in this project.

TIMELINE | STUDY MANAGEMENT | RESEARCHERS AND PARTICIPANTS | CONNECTIONS



Profile

Home > [DCM210] AI Workshop: Customized your thingCV > Object graph generator

id: 7730 EXISTING PUBLIC MIT

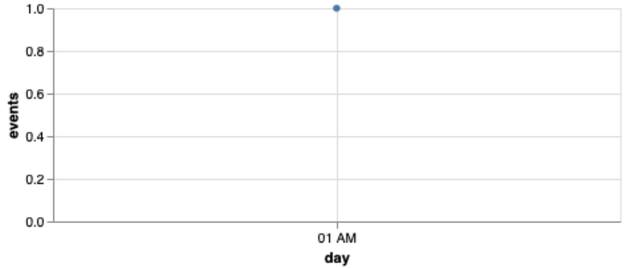
OBJECT GRAPH GENERATOR

2023-03-01 — 2024-03-01

generate object graph based on data collected from DF

INFO

License: MIT



VIEW DATA

DOWNLOAD

UPLOAD FILE(S)

DATASET FILES

File name	Description	Uploaded	
generate object_graph.gg	object graph generator	Mar 12 at 13:30	edit | delete

<< back to dataset page

Notebook: starBoardTest.gg

Edit your notebook below, don't forget to save your changes. You can access your datasets with a bit of JavaScript or Python, depending on the notebook cell.

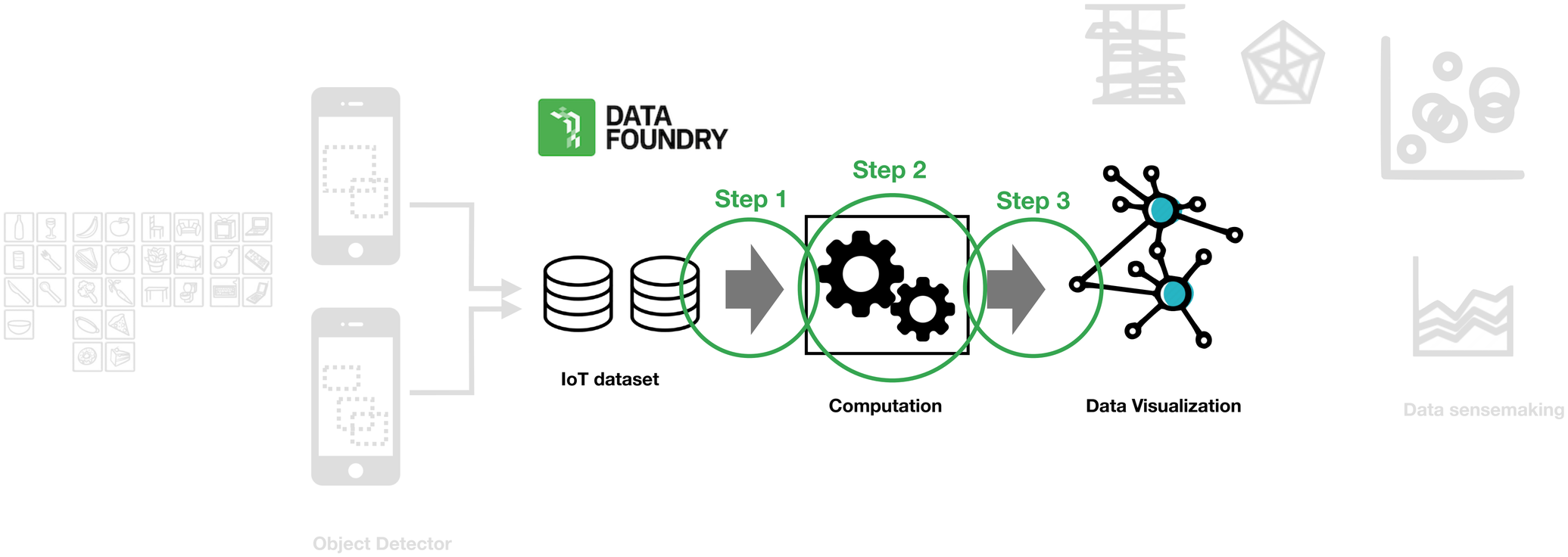
Save notebook

JAVASCRIPT

```javascript
let what = {
    test: true
}

console.log(what)
```

PYTHON

```python
globalWhat = {123}

print(globalWhat)

```



(1) **Read data** from Data foundry
(2) **Build object network** based on collected data through similarity calculation (starboard on DF)
(3) **Integrate all components** (hosting a web-based application on DF)

Notebook: generate object_graph.gg

https://data.id.tue.nl/datasets/existing/edit/7730/1/generate%20object_graph.gg

Edit your notebook below, don't forget to save your changes. You can access your datasets with a bit of JavaScript or Python, depending on the notebook cell.

Save notebook

1. Read data from Data foundry

JAVASCRIPT

```
//read data from DF
data_url = "https://data.id.tue.nl/datasets/downloadPublic/json/M1pxNVIrWU8zMUg4L21zN0JwbU1WNjBxUmVLa0ZiRnVwVlc5L2
var jdata = await fetch(data_url);
var jsonData = await jdata.json();

var object_data = [];
  for(d in jsonData){
    if(jsonData[d].detections != ""){
      object_data.push(jsonData[d].detections.split(","));
    }
  }
console.log(object_data);
```

```
▼(29) [Array(1), Array(1), Array(1), Array(1), Array(1), Array(1), Array(1), Array(1), Array(1), Array(1), …]
  ▶0: Array(1)
  ▶1: Array(1)
  ▶2: Array(1)
  ▶3: Array(1)
  ▼4: Array(1)
     0: "tv"
  ▶5: Array(1)
  ▶6: Array(1)
  ▶7: Array(1)
  ▶8: Array(1)
  ▶9: Array(1)
  ▶10: Array(1)
  ▶11: Array(1)
  ▶12: Array(1)
  ▶13: Array(1)
  ▶14: Array(1)
```

```
obj_img_map = {}
object_data.forEach((obj_list, indx) => {
  obj_list.forEach(obj => {
    if(obj in obj_img_map){
      obj_img_map[obj].push(indx)
    }else{
      obj_img_map[obj] = []
      obj_img_map[obj].push(indx)
    }
  })

})
console.log(obj_img_map);
```

```
▼{tv: Array(20), laptop: Array(6), person: Array(12), mouse: Array(3), car: Array(1)}
  ▶tv: Array(20)
  ▼laptop: Array(6)
     0: 17
     1: 18
     2: 19
     3: 20
     4: 23
     5: 24
  ▶person: Array(12)
  ▶mouse: Array(3)
  ▶car: Array(1)
```

2. Create index map for each object

3. Calculate similarity score for each link

```
object_net = {}

for(obj_1 in obj_img_map){

  if(!(obj_1 in object_net)){
    object_net[obj_1] = {};
  }

  for(obj_2 in obj_img_map){
    if(obj_2 == obj_1){
      object_net[obj_1][obj_2] = 1.0;
      continue;
    }

    if(!(obj_2 in object_net[obj_1])){
      let intersection = obj_img_map[obj_1].filter(x => obj_img_map[obj_2].includes(x));
      let union = [...obj_img_map[obj_1], ...obj_img_map[obj_2]];

      score = (new Set(intersection)).size*1.0 / (new Set(union)).size;
    }
    object_net[obj_1][obj_2] = score;
  }
}

for(obj in object_net){
  console.log(object_net[obj]);
}
```

```
▼{tv: 1, laptop: 0.13043478260869565, person: 0, mouse: 0, car: 0}
    tv: 1
    laptop: 0.13043478260869565
    person: 0
    mouse: 0
    car: 0
  ▶{tv: 0.13043478260869565, laptop: 1, person: 0.125, mouse: 0, car: 0}
  ▶{tv: 0, laptop: 0.125, person: 1, mouse: 0, car: 0}
  ▶{tv: 0, laptop: 0, person: 0, mouse: 1, car: 0}
```

Notebook: generate object_graph.gg

Edit your notebook below, don't forget to save your changes. You can access your datasets with a bit of JavaScript or Python, depending on the notebook cell.

Save notebook

```javascript
//read data from DF
data_url = "https://data.id.tue.nl/datasets/downloadPublic/json/M1pxNVIrMU8zMUg4L21zN0JwbU1WNjBxUnVLa0ZiRnVwVlc5Li
var jdata = await fetch(data_url);
var jsonData = await jdata.json();

var object_data = [];
  for(d in jsonData){
    if(jsonData[d].detections != ""){
      object_data.push(jsonData[d].detections.split(","));
    }
  }
console.log(object_data);
```

```javascript
//create a fake data to test out the algorithm
data = [
  ["obj1", "obj2", "obj3", "obj1"],
  ["obj1", "obj3", "obj3"],
  ["obj3", "obj4"],
  ["obj2"],
  ["obj1", "obj5"],
  ["obj3"],
  ["obj5", "obj2"],
  ["obj1", "obj4", "obj6"],
  ["obj2", "obj6", "obj5", "obj1"],
  ["obj4", "obj5"]
]
```

```javascript
obj_img_map = {}
data.forEach((obj_list, indx) => {
  obj_list.forEach(obj => {
    if(obj in obj_img_map){
      obj_img_map[obj].push(indx)
    }else{
      obj_img_map[obj] = []
      obj_img_map[obj].push(indx)
    }
  })

})
console.log(obj_img_map);
```

```javascript
object_net = {}

for(obj_1 in obj_img_map){

  if(!(obj_1 in object_net)){
    object_net[obj_1] = {};
  }

  for(obj_2 in obj_img_map){
    if(obj_2 == obj_1){
      object_net[obj_1][obj_2] = 1.0;
      continue;
    }

    if(!(obj_2 in object_net[obj_1])){
      let intersection = obj_img_map[obj_1].filter(x => obj_img_map[obj_2].includes(x));
```

4. Copy the code from starboard to the website

testing notebook environment — production site

Home > [DCM210] AI Workshop: Customized your thingCV > ThingCV > Edit file

EDIT FILE

Edit contents of 'index.html' below, press 'SAVE' to save your changes.

```
    .attr("r", 5);

var root = {};

var readDataFromDF = async function() {
  data_url = "https://data.id.tue.nl/datasets/downloadPublic/json/NFFPcDVjdzdSbjV0VFl1SXI4aUI4Y2REOFl4cHZIRkdNTk5rRlRucHoyST0=";
  var jdata = await fetch(data_url);

  var jsonData = await jdata.json();
  var object_data = [];

  for(d in jsonData){
    if(jsonData[d].detections != ""){
      object_data.push(jsonData[d].detections.split(","));
    }
  }

  graph = calculateGraph(object_data);
  return graph;
};

function calculateGraph(data){
  obj_img_map = {}
  data.forEach((obj_list, indx) => {
    obj_list.forEach(obj => {
        if(obj in obj_img_map){
          obj_img_map[obj].push(indx)
        }else{
          obj_img_map[obj] = []
          obj_img_map[obj].push(indx)
        }
```

SAVE CANCEL

Session II: Build your LLMs model on DF

Introduction

Chat models take a series of messages as input, and return a model-generated message as output.

Although the chat format is designed to make multi-turn conversations easy, it's just as useful for single-turn tasks without any conversations (such as those previously served by instruction following models like `text-davinci-003`).

An example API call looks as follows:

```
# Note: you need to be using OpenAI Python v0.27.0 for the code below to
import openai

openai.ChatCompletion.create(
  model="gpt-3.5-turbo",
  messages=[
        {"role": "system", "content": "You are a helpful assistant."},
        {"role": "user", "content": "Who won the world series in 2020?"},
        {"role": "assistant", "content": "The Los Angeles Dodgers won the World
        {"role": "user", "content": "Where was it played?"}
    ]
)
```

The main input is the messages parameter. Messages must be an array of message objects, where each object has a role (either "system", "user", or "assistant") and content (the content of the message). Conversations can be as short as 1 message or fill many pages.

Typically, a conversation is formatted with a system message first, followed by alternating user and assistant messages.

The system message helps set the behavior of the assistant. In the example above, the assistant was instructed with "You are a helpful assistant."

The user messages help instruct the assistant. They can be generated by the end users of an application, or set by a developer as an instruction.

The assistant messages help store prior responses. They can also be written by a developer to help give examples of desired behavior.

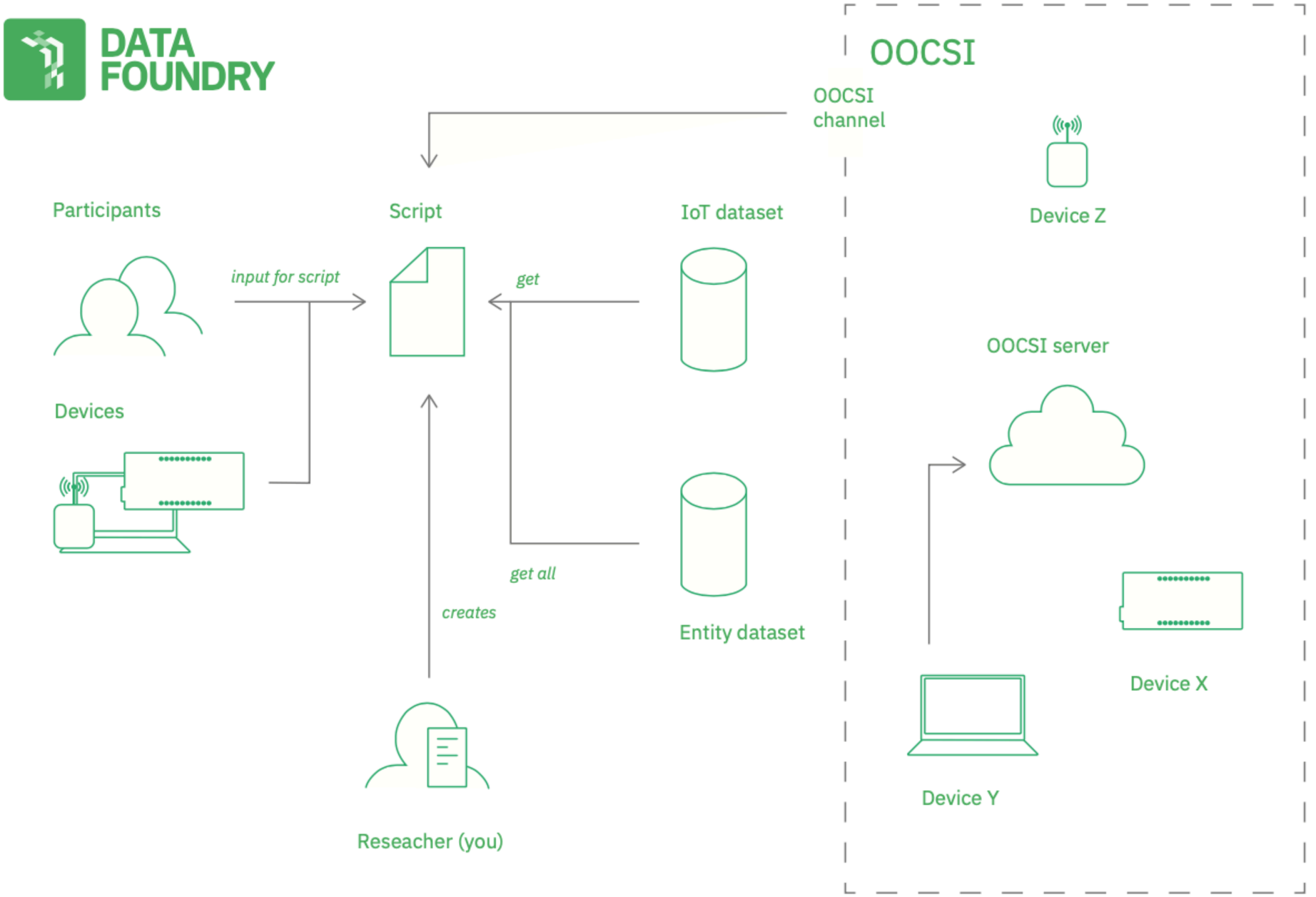
Including the conversation history helps when user instructions refer to prior messages. In the example above, the user's final question of "Where was it played?" only makes sense in the context of the prior messages about the World Series of 2020. Because the models have no memory of past requests, all relevant information must be supplied via the conversation. If a conversation cannot fit within the model's token limit, it will need to be shortened in some way.

Response format

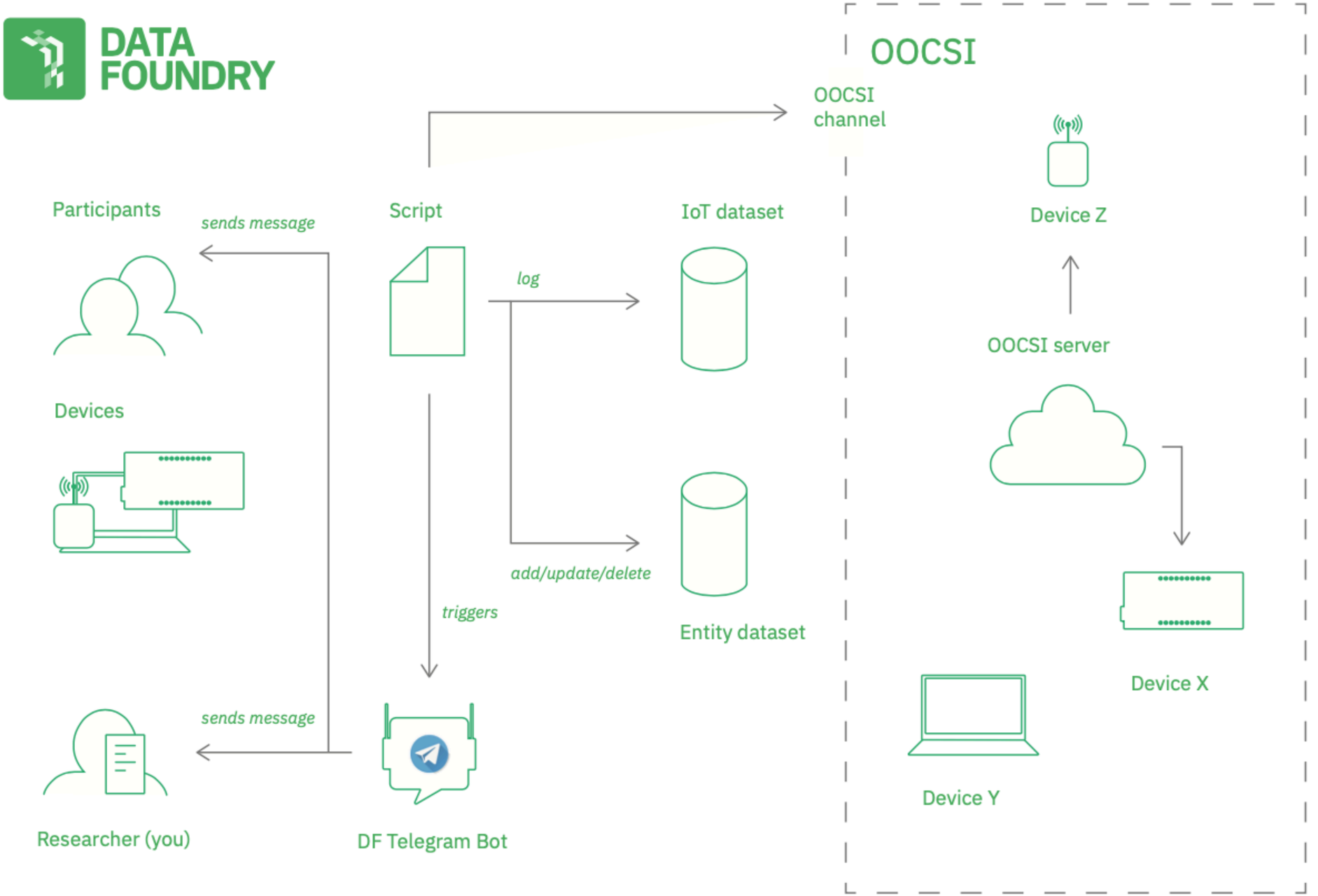
An example API response looks as follows:

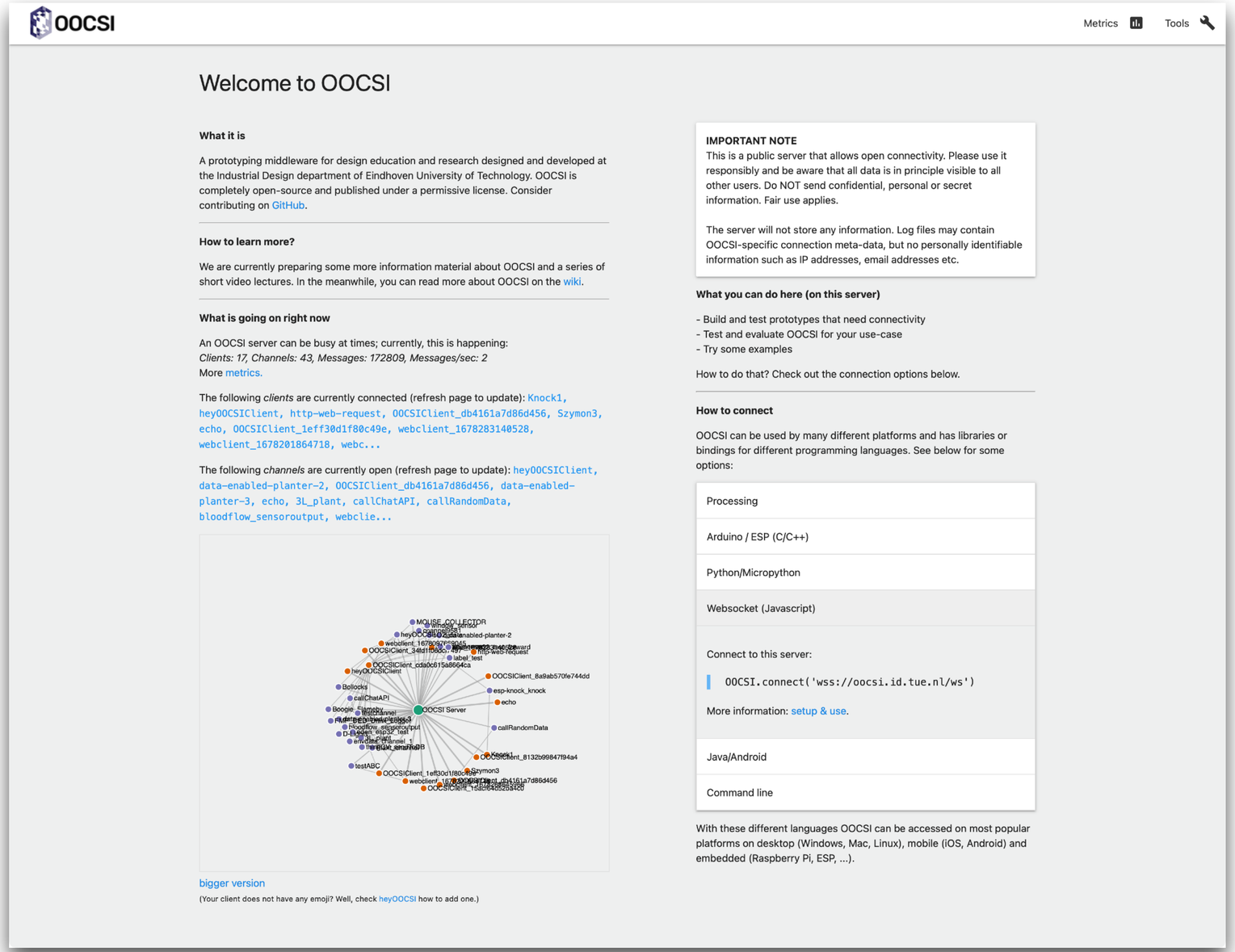
https://platform.openai.com/docs/guides/chat
https://platform.openai.com/docs/api-reference/chat

Script Input



Script Output





OOCSI

OOCSI is **a prototyping "middleware"** for designing distributed products and it is targeted mainly at Industrial Design and Computer Science education.

OOCSI supports multiple client platforms that allow **prototyping connected products and systemic designs with various heterogeneous components** from **embedded** to **mobile** to the **cloud**.

https://oocsi.id.tue.nl/

https://github.com/iddi/oocsi

https://oocsi.id.tue.nl/

OOCSI

callChatAI

DF Script

DATA FOUNDRY

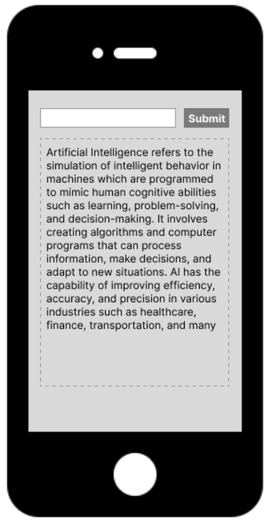
script input

input

script output

OpenAI

- Text completion
- Chat completion
- Moderation



output

returnChatAI

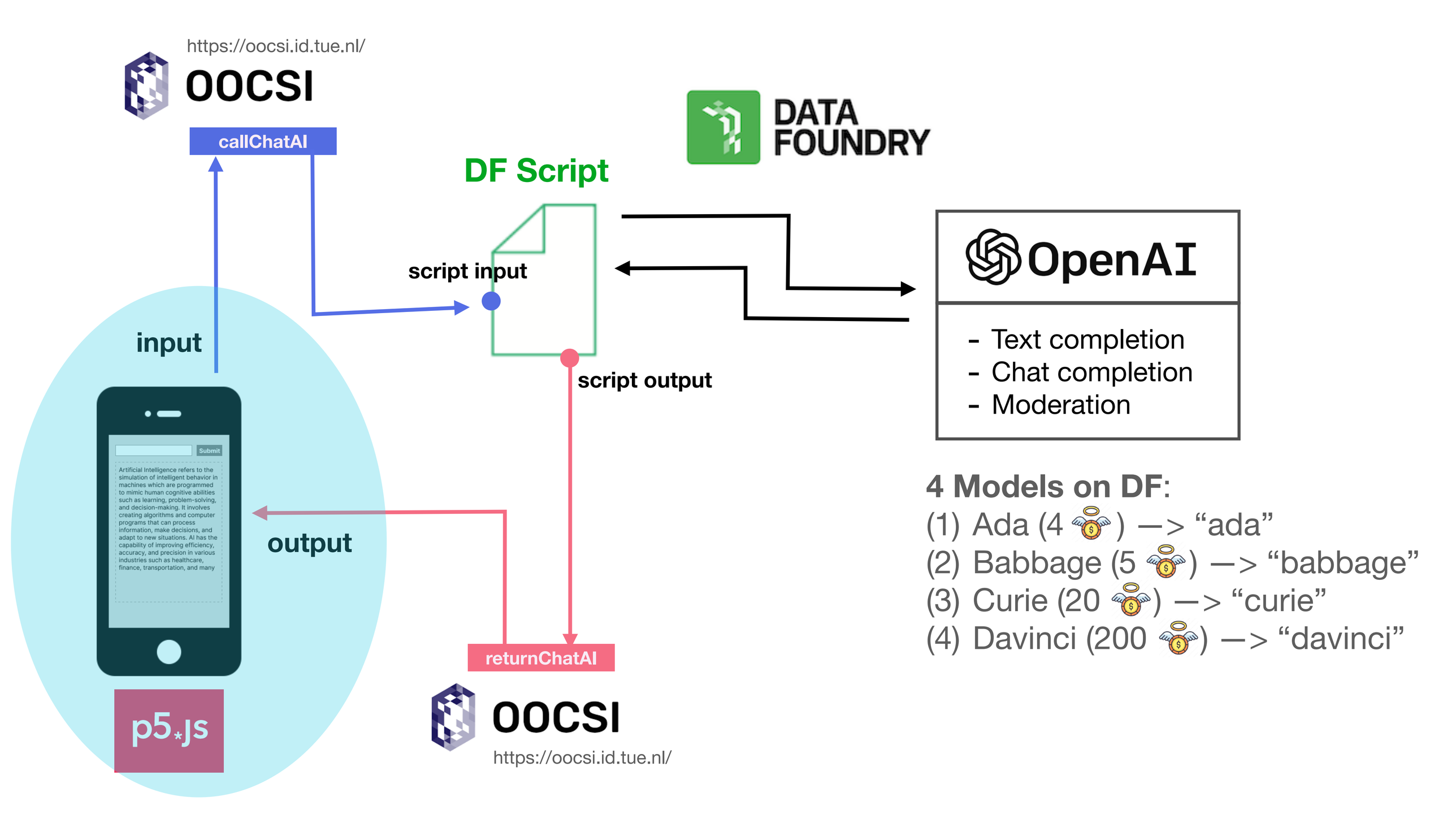
p5.js

OOCSI

https://oocsi.id.tue.nl/

4 Models on DF:

(1) Ada (4 💰) —> “ada”
(2) Babbage (5 💰) —> “babbage”
(3) Curie (20 💰) —> “curie”
(4) Davinci (200 💰) —> “davinci”



4 Models on DF:

(1) Ada (4 💸) —> “ada”
(2) Babbage (5 💸) —> “babbage”
(3) Curie (20 💸) —> “curie”
(4) Davinci (200 💸) —> “davinci”

p5* File Edit Sketch Help

Auto-refresh Test OpenAPI on DF by janetyc

sketch.js Saved: just now

```
  //create a button
  button = createButton("submit");
  button.position(220,10);
  button.mouseClicked(buttonEvent);

  //create a output div
  output = createDiv();
  output.position(10,30);

  //connect to DF
  OOCSI.connect('wss://oocsi.id.tue.nl/ws');

  // this is where you receive the data back from the DF script
  // that means the DF script needs to use DF.oocsi('returnChatAPI', {text: "gpt generated content"})
  OOCSI.subscribe("returnChatAPI", function(e) {
    print(e)
    if(e.error){
      output.html("ask:"+input.value()+"<br>ans:"+e.error);
    }else{
      output.html("ask:"+input.value()+"<br>ans:"+e.data.text);
    }
    input.value("");
  });

}

function draw() {
  background(220);
}

function buttonEvent() {
  print("send data to DF");
  print(input.value());
  var data = {
    "input": input.value()
  };

  OOCSI.send("callChatAPI", data);
```

Preview

Console Clear

https://editor.p5js.org/janetyc/sketches/L92YsBtK7

Step 1: Create a visual layout of your interface

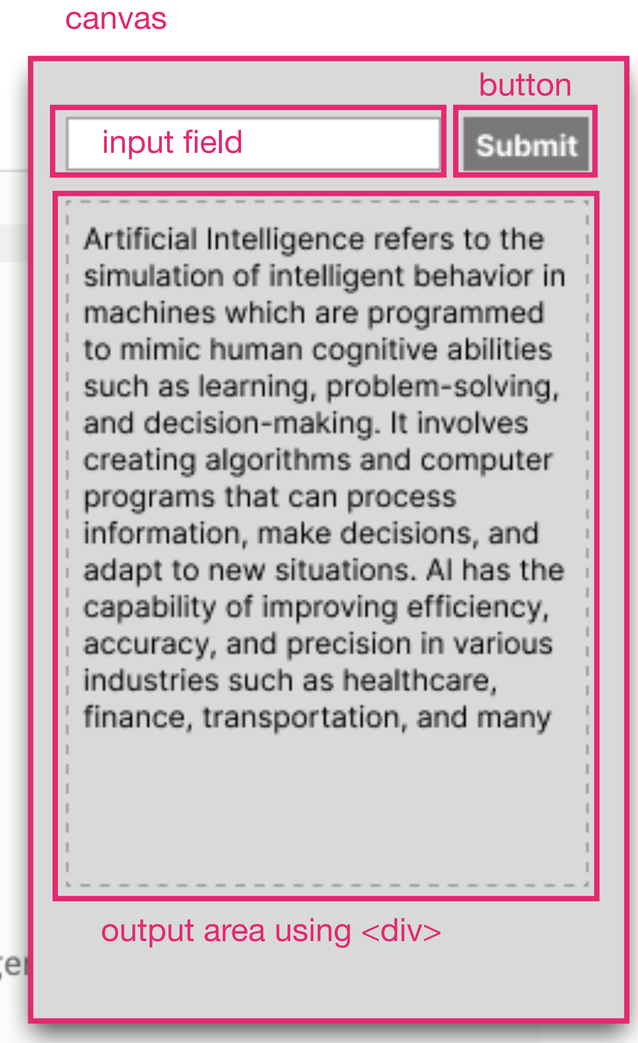
```
function setup() {
  createCanvas(400, 400);
  input = createInput();
  input.size(200, 10);
  input.position(10, 10);

  //create a button
  button = createButton("submit");
  button.position(220,10);
  button.mouseClicked(buttonEvent);

  //create a output div
  output = createDiv();
  output.position(10,30);

  //connect to DF
  OOCSI.connect('wss://oocsi.id.tue.nl/ws');

  // this is where you receive the data back from the DF script
  // that means the DF script needs to use DF.oocsi('returnChatAPI', {text: "gpt ge
  OOCSI.subscribe("returnChatAPI", function(e) {
    print(e)
    if(e.error){
      output.html("ask:"+input.value()+"<br>ans:"+e.error);
    }else{
      output.html("ask:"+input.value()+"<br>ans:"+e.data.text);
    }
    input.value("");
  });

}
```

create a display canvas

create an input field

create a button

create a output area



** : the content division HTML element is the general container for flow content**

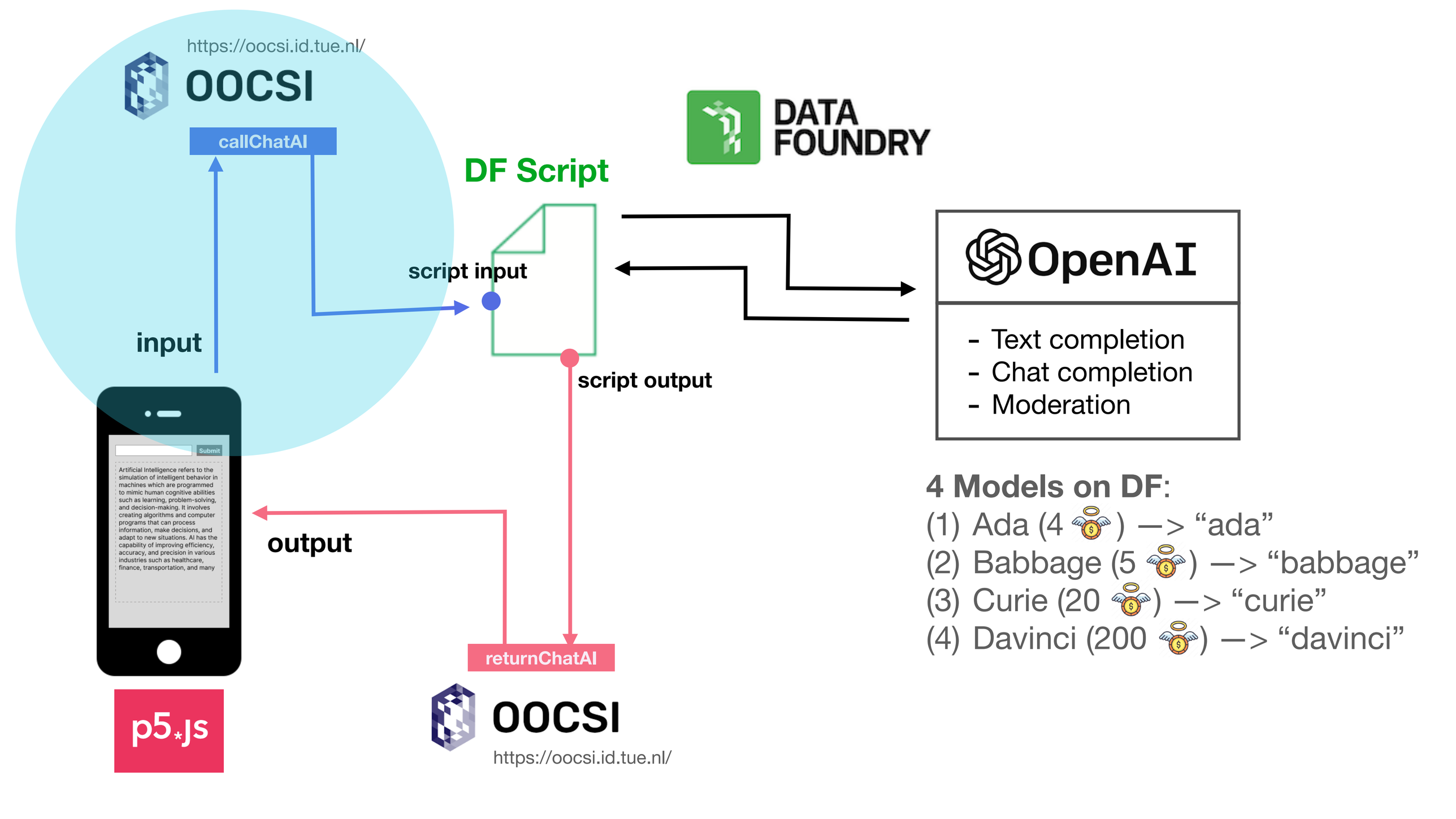
Step 2: add a mouse event

```
function setup() {
  createCanvas(400, 400);
  input = createInput();
  input.size(200, 10);
  input.position(10, 10);

  //create a button
  button = createButton("submit");
  button.position(220,10);
  button.mouseClicked(buttonEvent);

  //create a output div
  output = createDiv();
  output.position(10,30);

  //connect to DF
  OOCSI.connect('wss://oocsi.id.tue.nl/ws');

  // this is where you receive the data back from the DF script
  // that means the DF script needs to use DF.oocsi('returnChatAPI', {text: "gpt generated content"})
  OOCSI.subscribe("returnChatAPI", function(e) {
    print(e)
    if(e.error){
      output.html("ask:"+input.value()+"<br>ans:"+e.error);
    }else{
      output.html("ask:"+input.value()+"<br>ans:"+e.data.text);
    }
    input.value("");
  });

}
```

add a mouseClicked event

When the button is clicked, then trigger this buttonEvent

```
function buttonEvent() {
  print("send data to DF");
  print(input.value());
  var data = {
    "input": input.value()
  };

  OOCSI.send("callChatAPI", data);

}
```


https://oocsi.id.tue.nl/
OOCSI
callChatAI
input
DF Script
script input
script output
DATA FOUNDRY
OpenAI
- Text completion
- Chat completion
- Moderation
Submit
Artificial Intelligence refers to the simulation of intelligent behavior in machines which are programmed to mimic human cognitive abilities such as learning, problem-solving, and decision-making. It involves creating algorithms and computer programs that can process information, make decisions, and adapt to new situations. AI has the capability of improving efficiency, accuracy, and precision in various industries such as healthcare, finance, transportation, and many
output
returnChatAI
p5.js
OOCSI
https://oocsi.id.tue.nl/
4 Models on DF:
(1) Ada (4) —> “ada”
(2) Babbage (5) —> “babbage”
(3) Curie (20) —> “curie”
(4) Davinci (200) —> “davinci”

Step 3: connect to OOCSI on DF

```
function setup() {
  createCanvas(400, 400);
  input = createInput();
  input.size(200, 10);
  input.position(10, 10);

  //create a button
  button = createButton("submit");
  button.position(220,10);
  button.mouseClicked(buttonEvent);

  //create a output div
  output = createDiv();
  output.position(10,30);

  //connect to DF
  OOCSI.connect('wss://oocsi.id.tue.nl/ws');

  // this is where you receive the data back from the DF script
  // that means the DF script needs to use DF.oocsi('returnChatAPI', {text: "gpt generated content"})
  OOCSI.subscribe("returnChatAPI", function(e) {
    print(e)
    if(e.error){
      output.html("ask:"+input.value()+"<br>ans:"+e.error);
    }else{
      output.html("ask:"+input.value()+"<br>ans:"+e.data.text);
    }
    input.value("");
  });

}
```

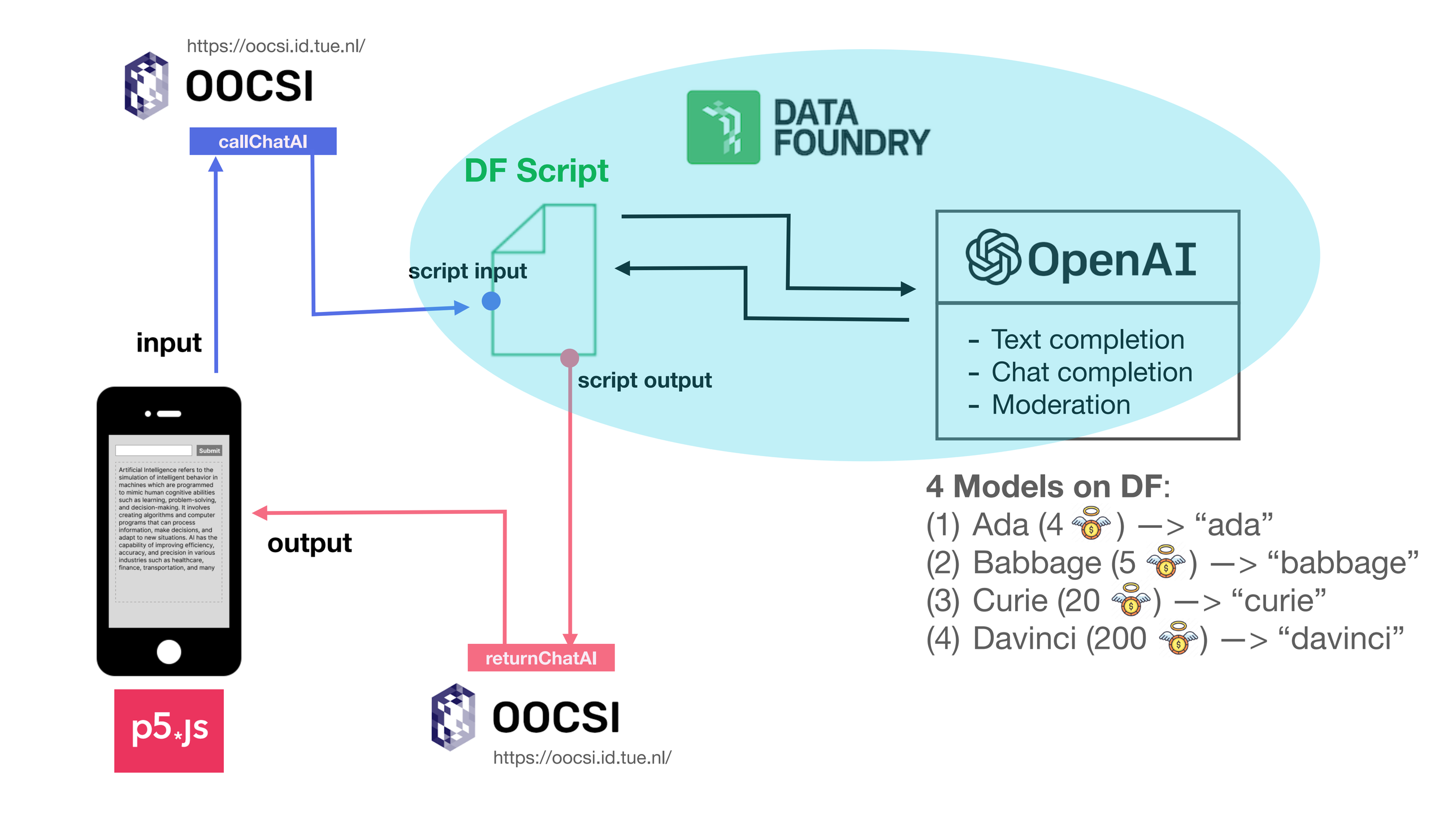
Step 4: send input data to DF via OOCSI channel

```
1  function setup() {
2    createCanvas(400, 400);
3    input = createInput();
4    input.size(200, 10);
5    input.position(10, 10);
6
7    //create a button
8    button = createButton("submit");
9    button.position(220,10);
10   button.mouseClicked(buttonEvent);
11
12   //create a output div
13   output = createDiv();
14   output.position(10,30);
15
16   //connect to DF
17   OOCSI.connect('wss://oocsi.id.tue.nl/ws');
18
19   // this is where you receive the data back from the DF script
20   // that means the DF script needs to use DF.oocsi('returnChatAPI', {text: "gpt generated content"})
21   OOCSI.subscribe("returnChatAPI", function(e) {
22     print(e)
23     if(e.error){
24       output.html("ask:"+input.value()+"<br>ans:"+e.error);
25     }else{
26       output.html("ask:"+input.value()+"<br>ans:"+e.data.text);
27     }
28     input.value("");
29   });
30
31 }
```

buttonClicked, then trigger

```
36
37 function buttonEvent() {
38   print("send data to DF");
39   print(input.value());
40   var data = {
41     "input": input.value()
42   };
43
44   OOCSI.send("callChatAPI", data);
45
46 }
47
```

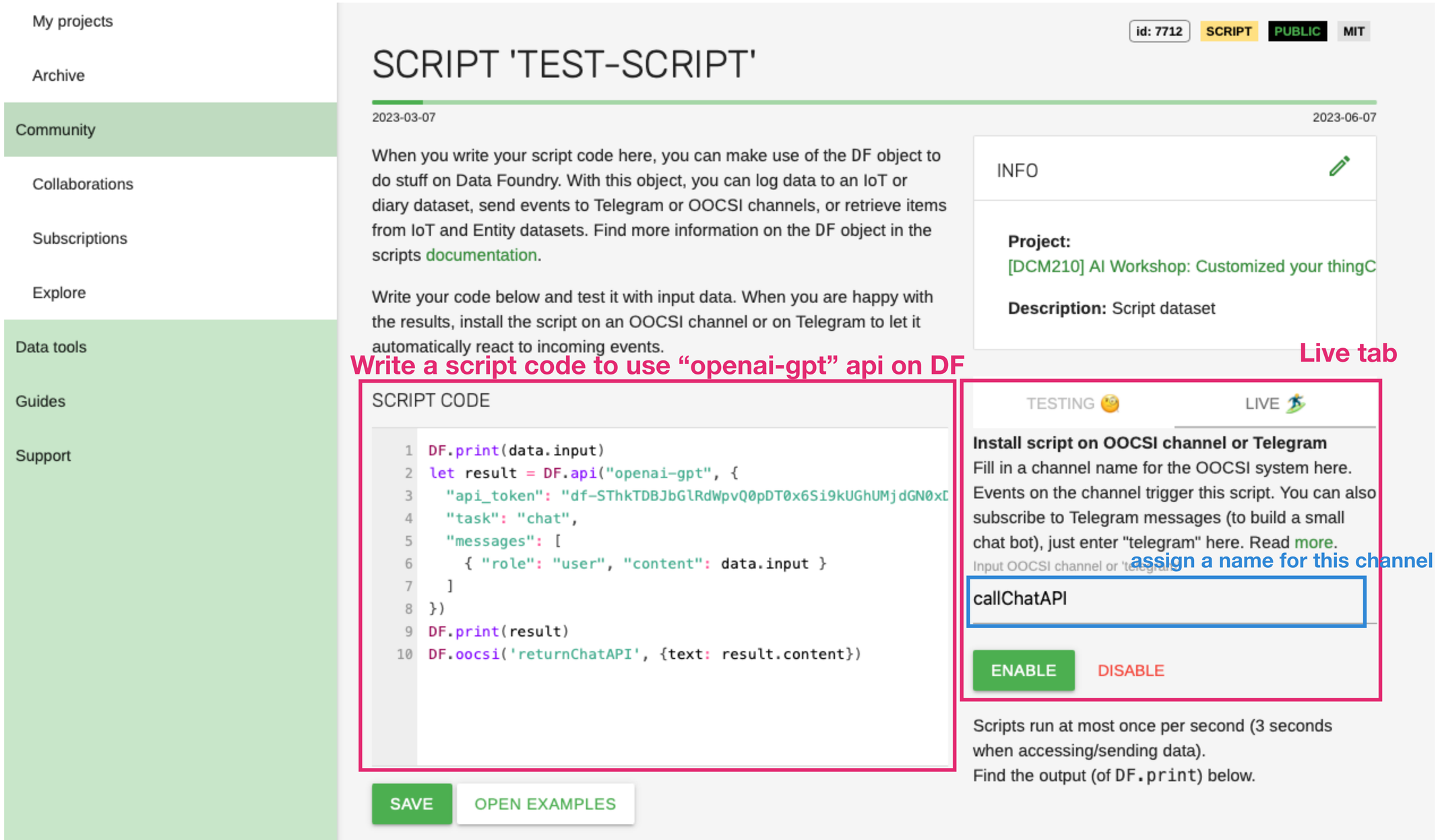
send data to “callChatAPI” channel



4 Models on DF:

(1) Ada (4 💸) —> “ada”
(2) Babbage (5 💸) —> “babbage”
(3) Curie (20 💸) —> “curie”
(4) Davinci (200 💸) —> “davinci”

Step 5: write a script to access OpenAI api on DF, and test it out



Step 5: write a script to access OpenAI api on DF, and test it out

SCRIPT CODE

```
1  DF.print(data.input)
2  let result = DF.api("openai-gpt", {
3    "api_token": "df-SThkTDBJbGlRdWpvQ0pDT0x6Si9kUGhUMjdGN0xDeU0rNnc
4    "task": "chat",
5    "messages": [
6      { "role": "user", "content": data.input }
7    ]
8  })
9  DF.print(result)
10 DF.oocsi('returnChatAPI', {text: result.content})
```

1. api_token

2. type of task

3. input data

SAVE OPEN EXAMPLES

4. Testing Tab: testing your script with input data

TESTING 🧐 LIVE 🏄

Test script with input data

Test your code above with data. Use the data object below to send data to your script.

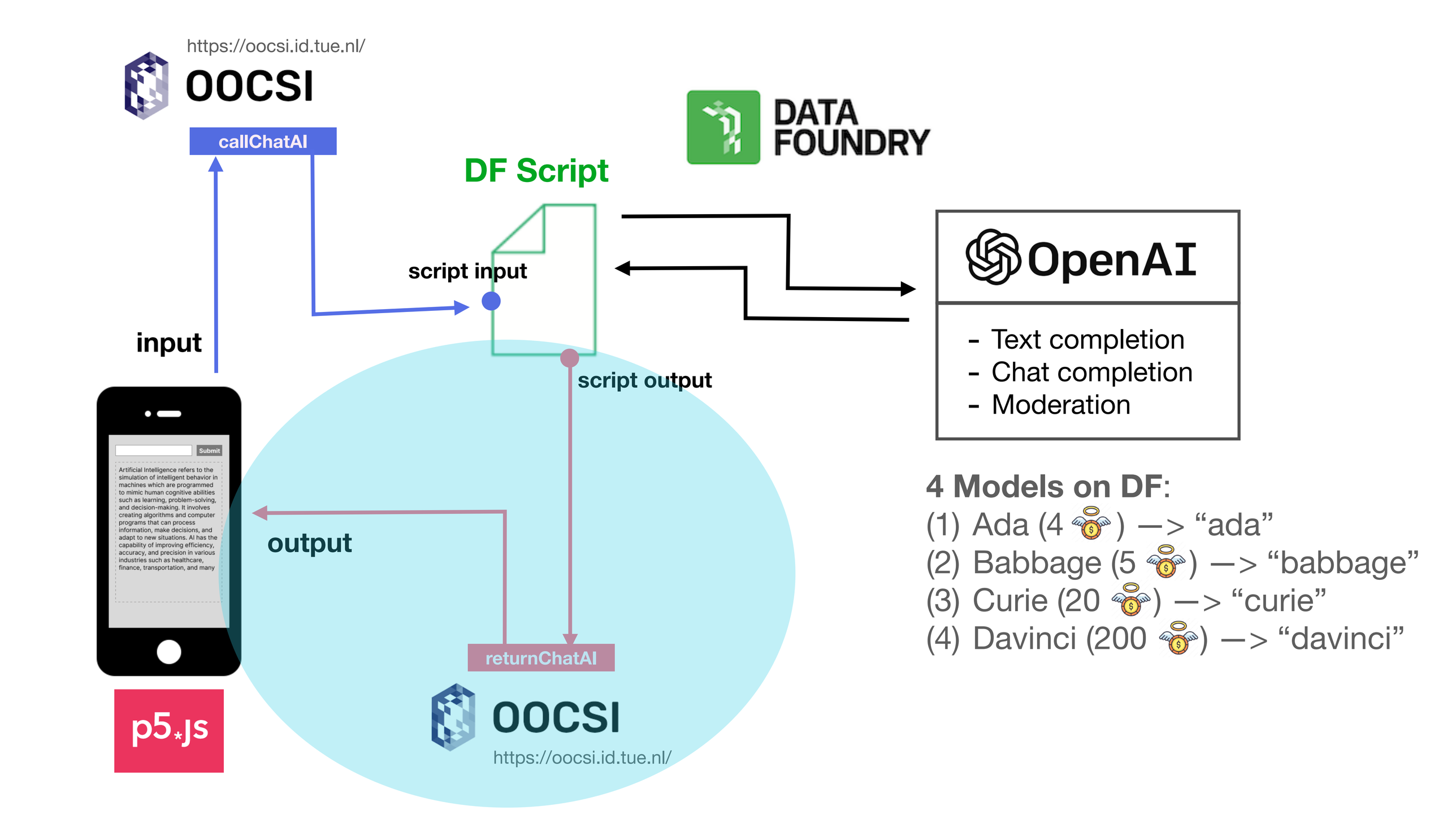
```
var data = {
  input: "What is AI?"
};
```

TEST WITH DATA

12:18:30 What is AI?

12:18:36 {"role":"assistant","content":"\n\nAI (Artificial Intelligence) is a branch of computer science that focuses on creating machines or software that can perform tasks that typically require human-level intelligence, such as visual perception, speech recognition, decision-making, and language translation. AI involves learning from experience, understanding natural language, and responding to

5. get output using “DF.print”

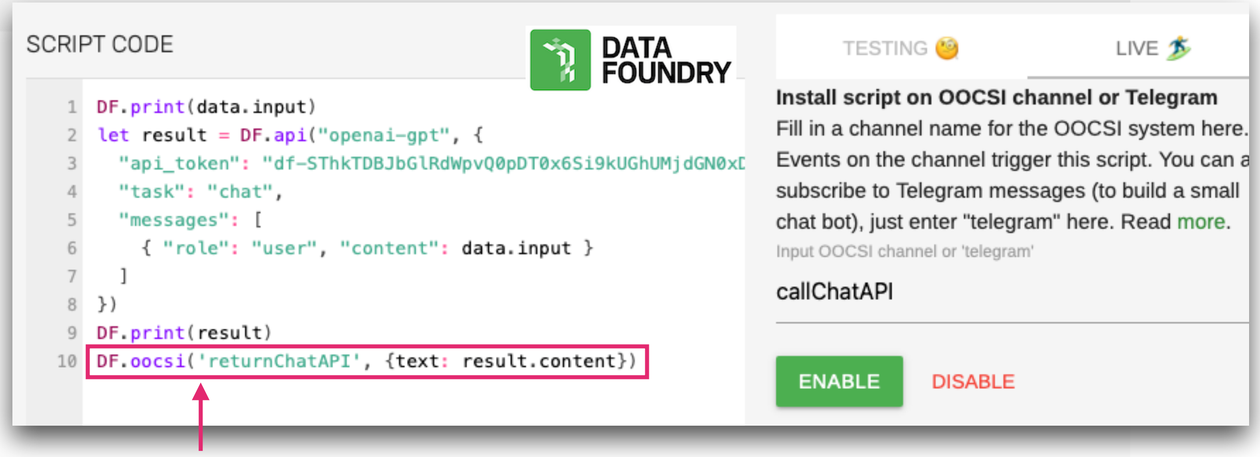


4 Models on DF:

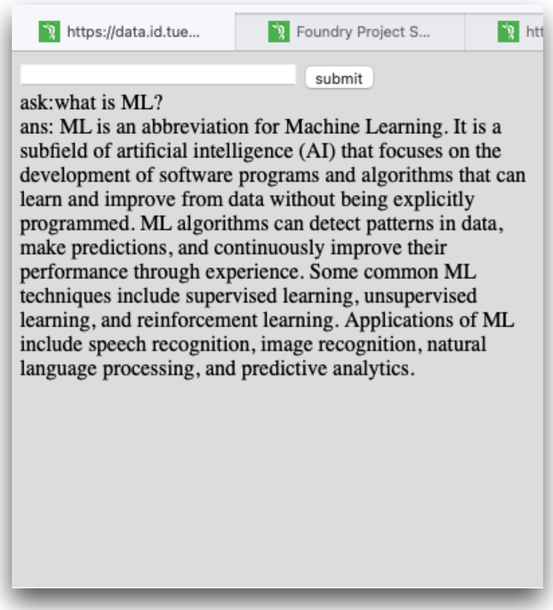
(1) Ada (4 💸) —> “ada”
(2) Babbage (5 💸) —> “babbage”
(3) Curie (20 💸) —> “curie”
(4) Davinci (200 💸) —> “davinci”

Step 6: subscribe a OOCSI channel to get output from ChatAPI

```
function setup() {
  createCanvas(400, 400);
  input = createInput();
  input.size(200, 10);
  input.position(10, 10);

  //create a button
  button = createButton("submit");
  button.position(220,10);
  button.mouseClicked(buttonEvent);

  //create a output div
  output = createDiv();
  output.position(10,30);

  //connect to DF
  OOCSI.connect('wss://oocsi.id.tue.nl/ws');

  // this is where you receive the data back from the DF script
  // that means the DF script needs to use DF.oocsi('returnChatAPI', {text: "gpt generated content"})
  OOCSI.subscribe("returnChatAPI", function(e) {
    print(e)
    if(e.error){
      output.html("ask:"+input.value()+"<br>ans:"+e.error);
    }else{
      output.html("ask:"+input.value()+"<br>ans:"+e.data.text);
    }
    input.value("");
  });

}
```



send data to “returnChatAPI” channel



https://oocsi.id.tue.nl/

OOCSI

callChatAI

DATA FOUNDRY

DF Script

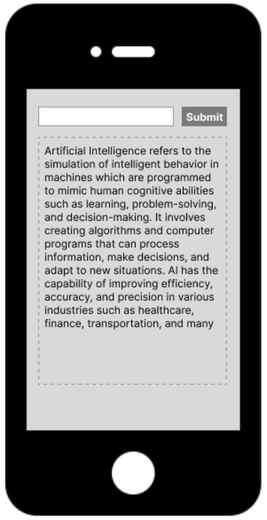
script input

OpenAI

- Text completion
- Chat completion
- Moderation

input

script output



output

returnChatAI

p5.js

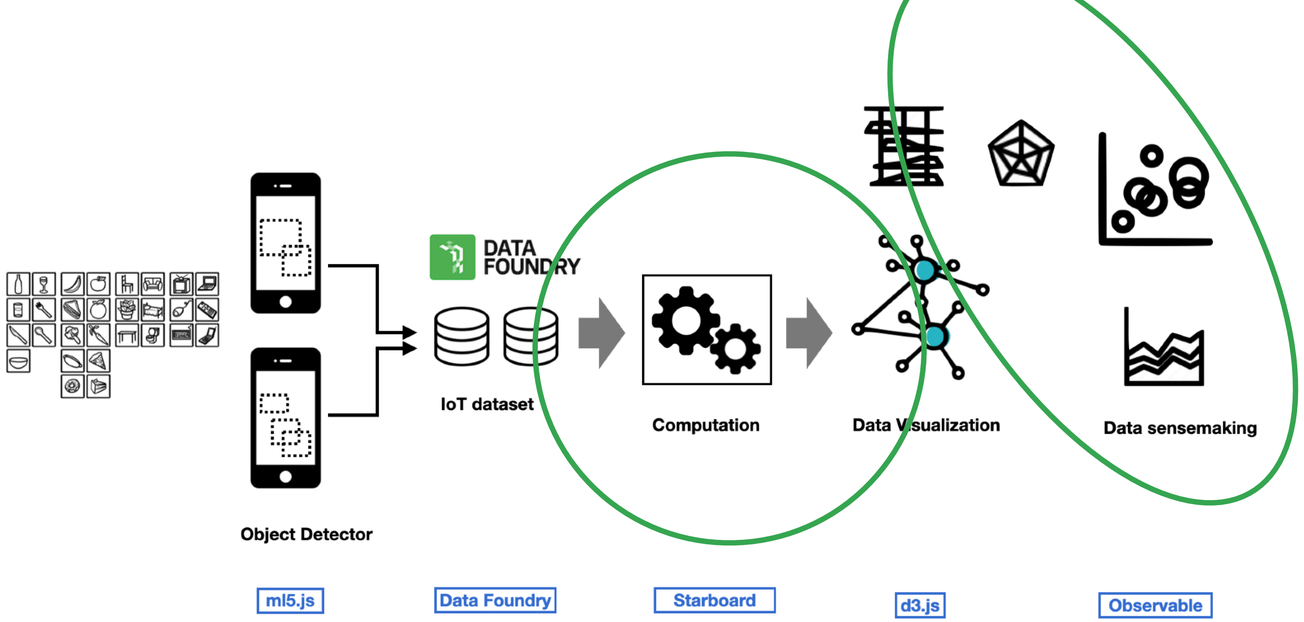
OOCSI

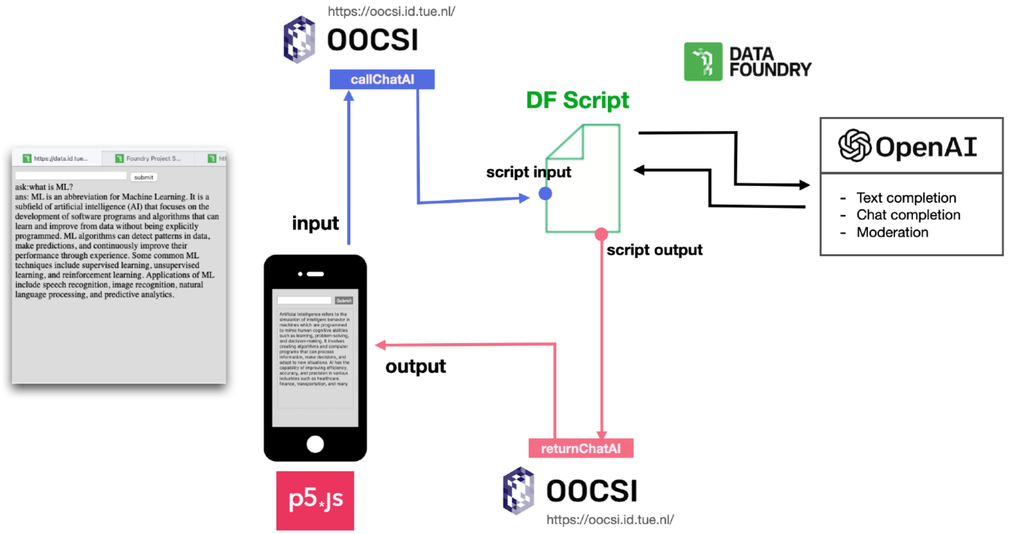
https://oocsi.id.tue.nl/

MDC Workshop (I): Advanced (2 hrs)

1. Build your thingCV from scratch (II): data, computation, algorithm **(30 mins)**
   - Foundation: integration of object hunter, network builder, and data visualizer (Data Foundry, starboard, d3.js)
   - Exercise 1: try out starboard,

2. Build your LLMs model on DF **(1.5 hrs)**
   - Foundation: OpenAI api in Data Foundry
   - Exercise 2: create an interface for interacting with OpenAI api





Thank You

Q&A