

# **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)

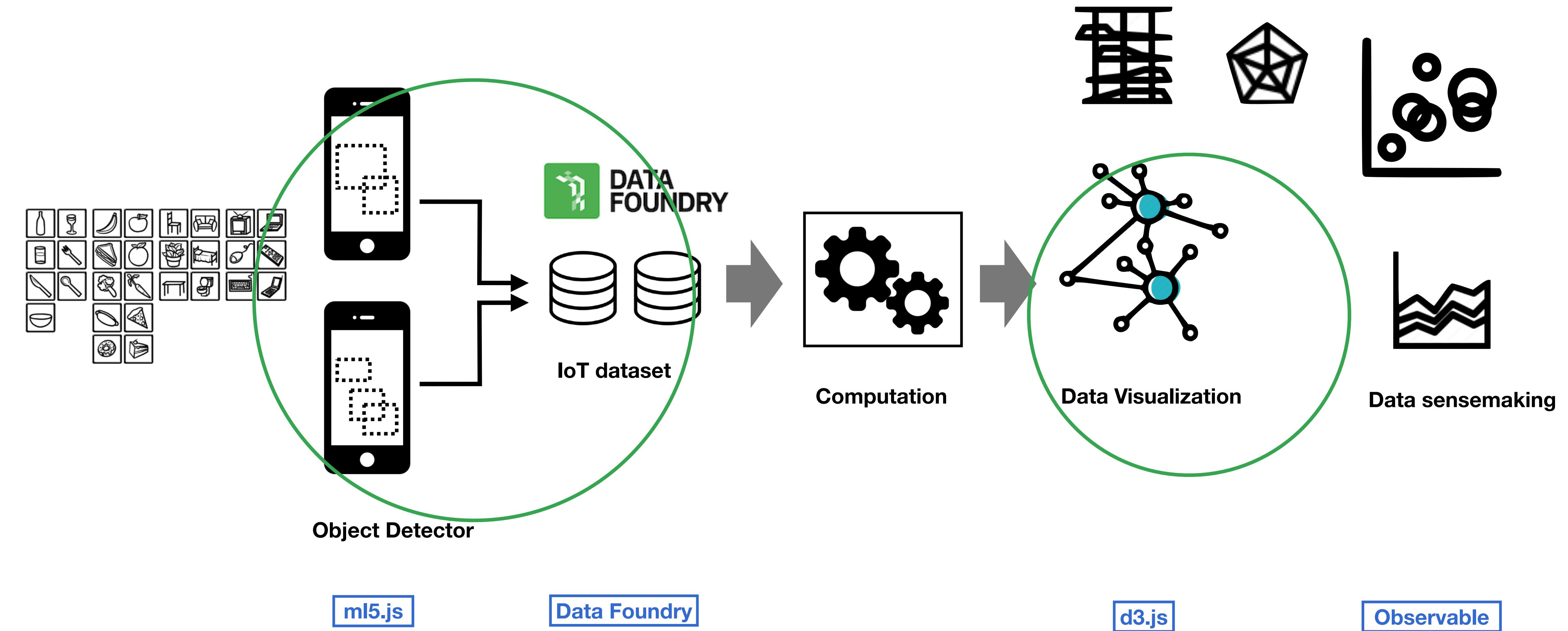
1. Introduction of thingCV (40 mins)
  - Introduction (10 mins)
  - Exercise 1: play with thingCV (20 mins)
  - Sharing (5-7 mins)
2. Collaborate with thingCV and generative AI (30 mins)
  - Introduction (5 mins)
  - Exercise 2: play with generative AI (20 mins)
  - Sharing (5 mins)
3. Build your thingCV 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)

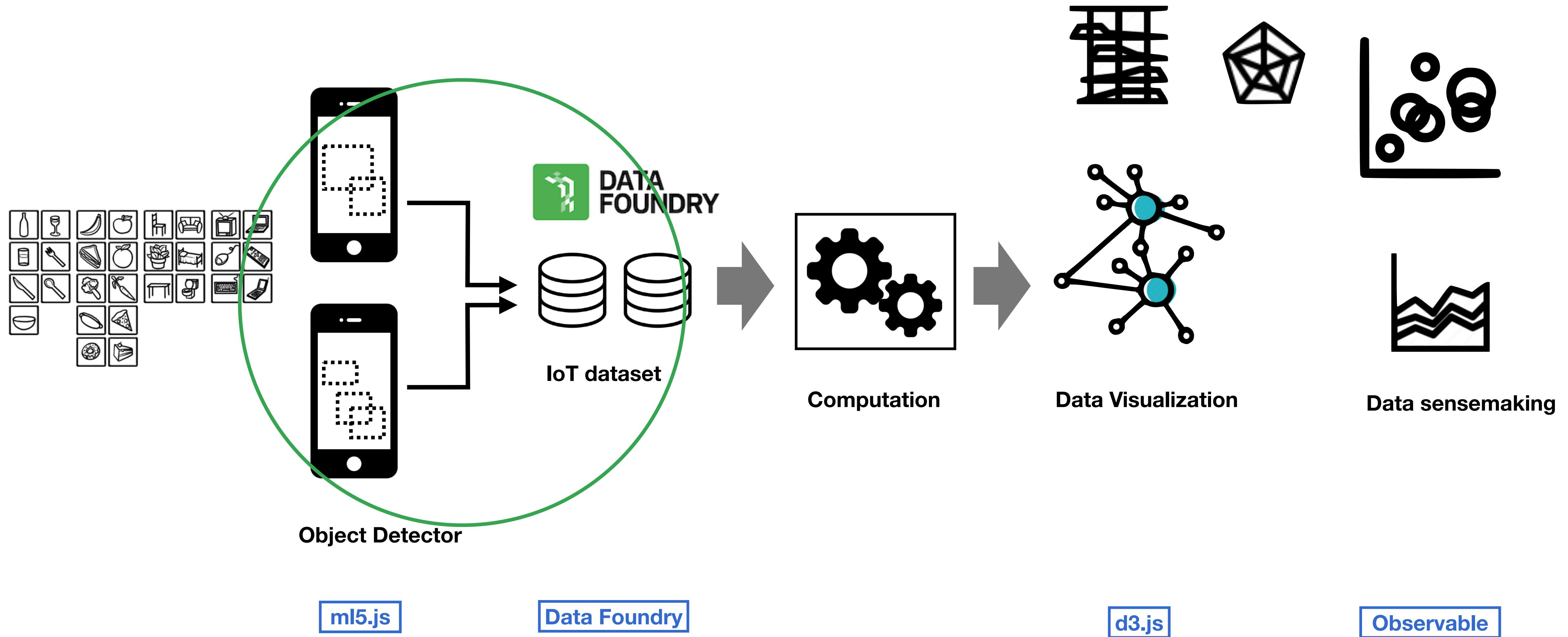
1. Build your thingCV from scratch (II): data, computation, algorithm (**30 mins**)
  - Foundation: integration of object hunter, network builder, and data visualizer (Data Foundry, starboard, d3.js)
  - Exercise 1: try out starboard,
2. Build your LLMs model (**1.5 hrs**)
  - Foundation: OpenAI api 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\* File Edit Sketch Help English Hello, janetycl!

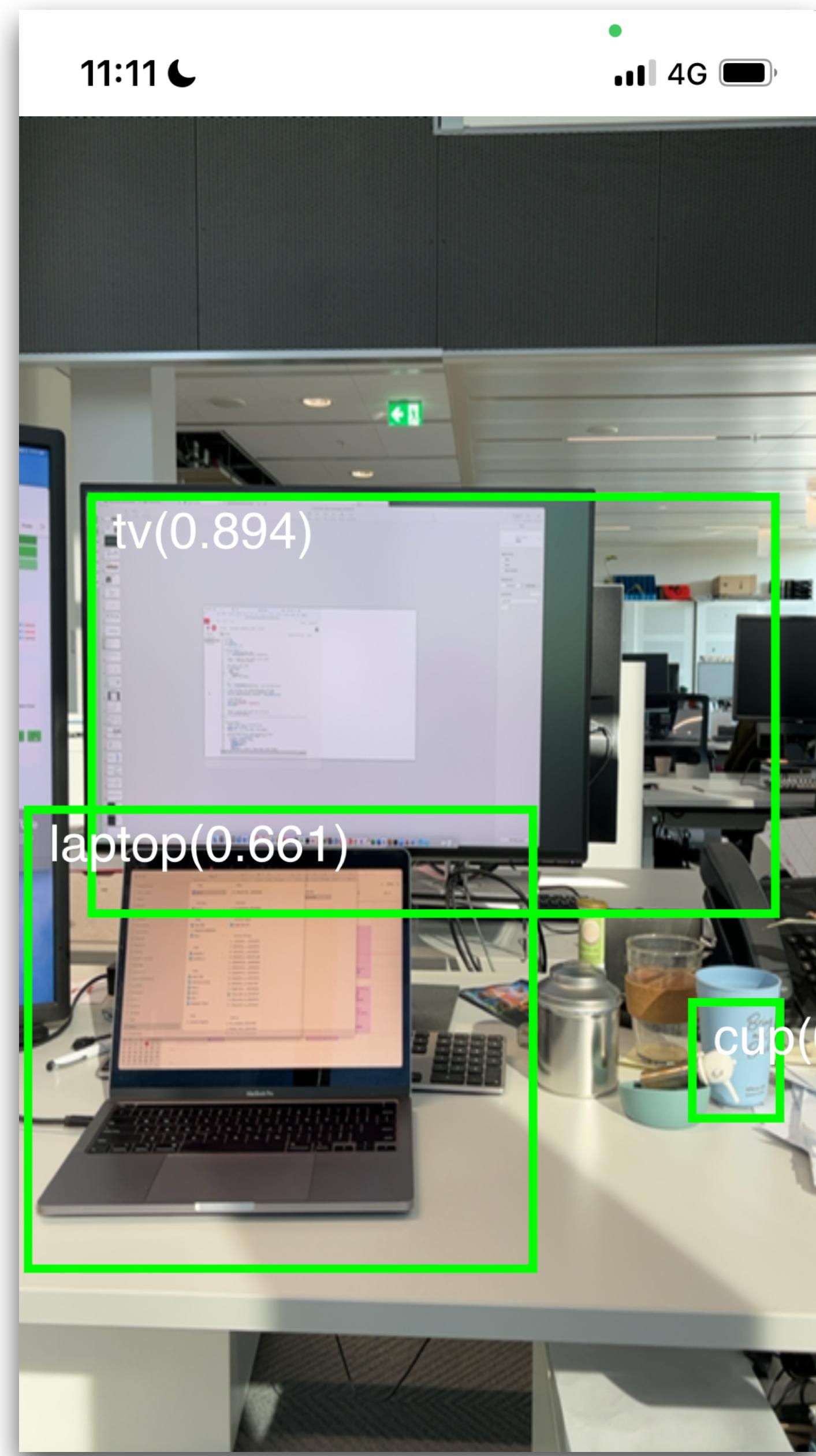
Auto-refresh MDC\_Workshop\_1\_ObjectDetector\_Webcam by janetyc

Sketch Files sketch.js

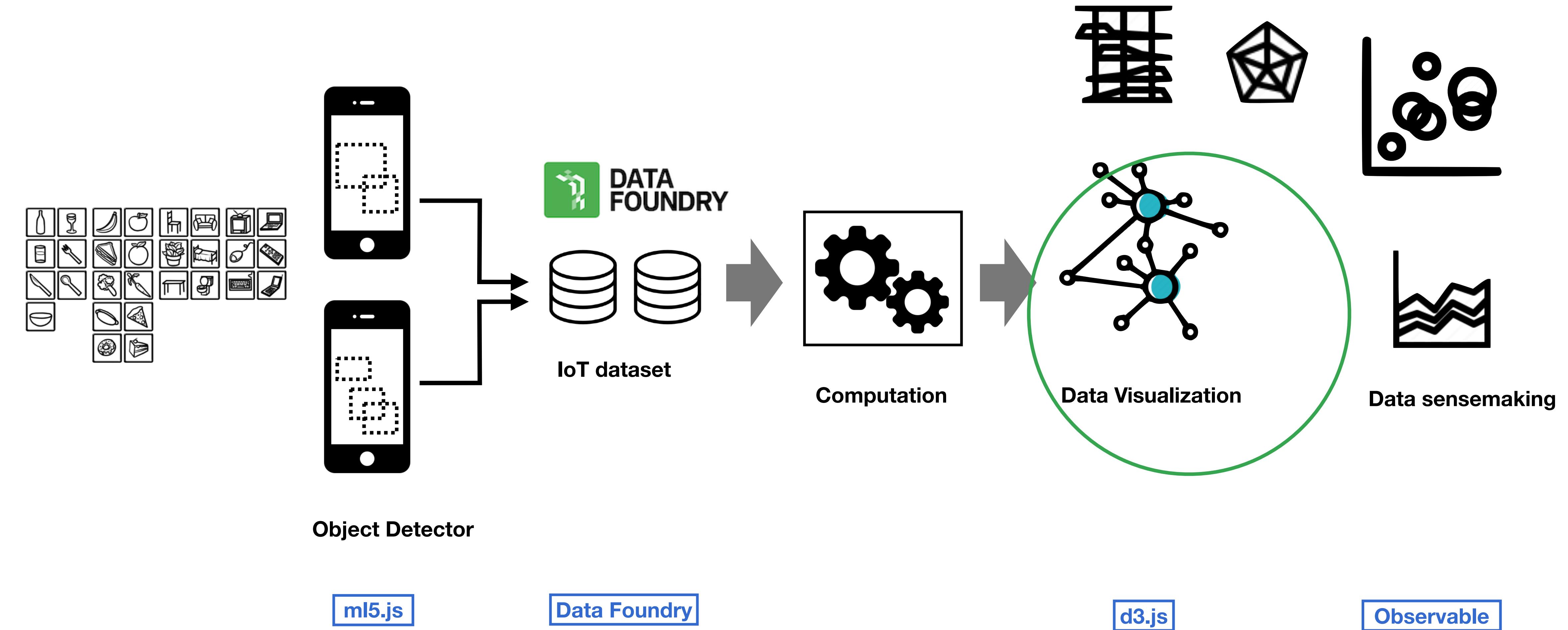
Saved: about 18 hours ago Preview

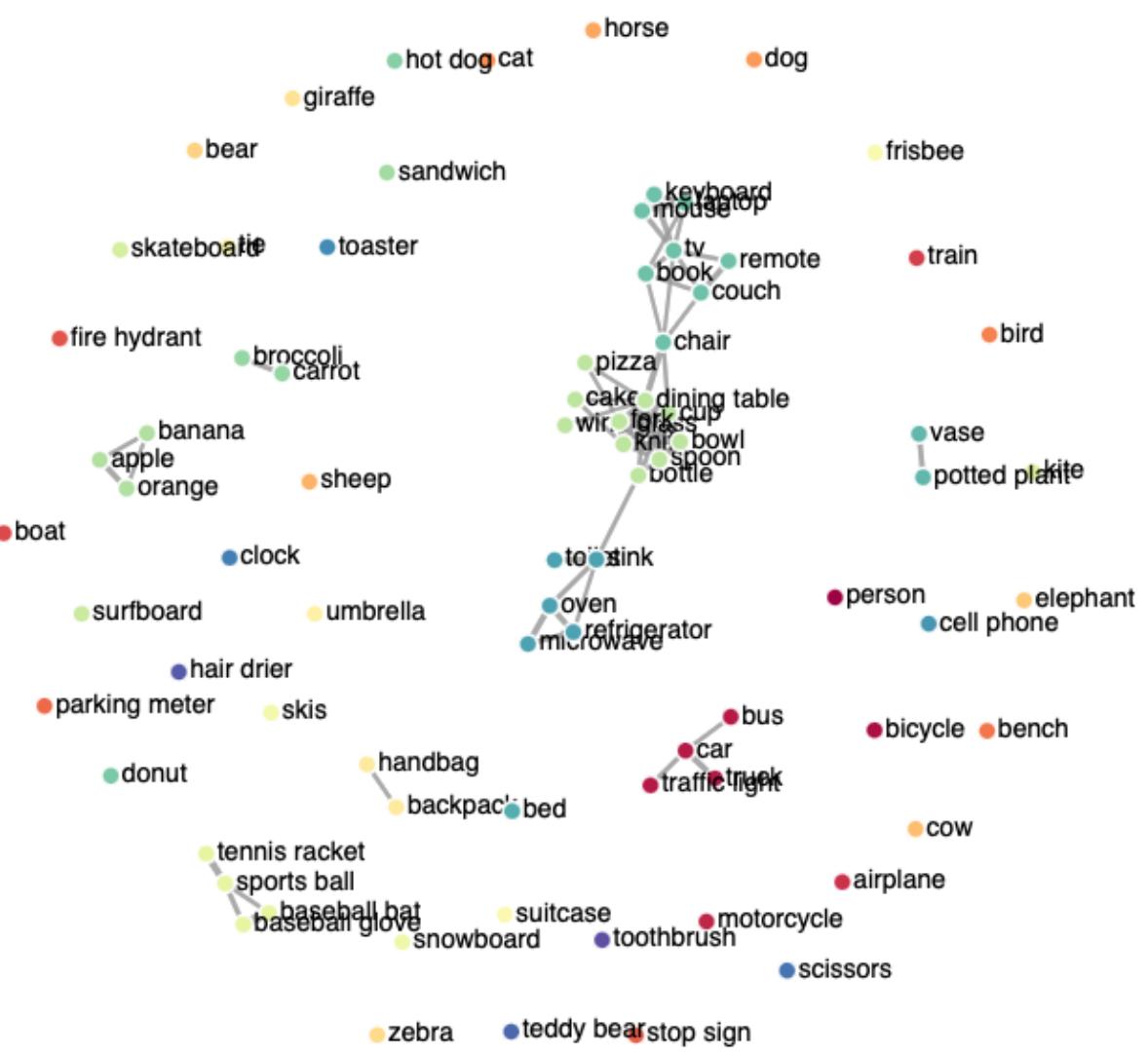
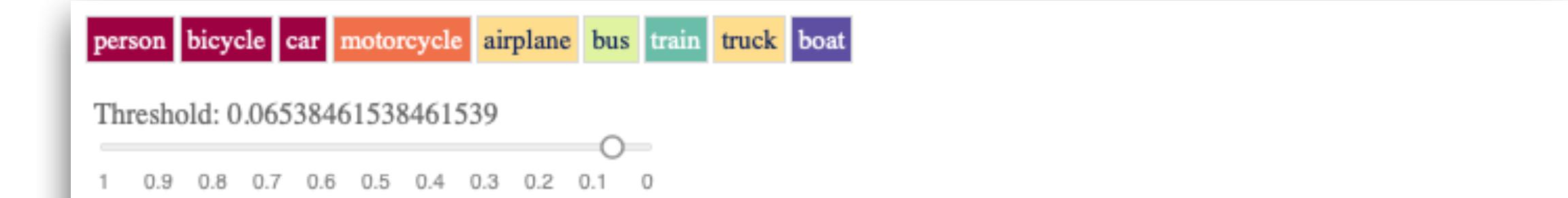
```
8
9 let cnv;
10 let video;
11 let detector;
12 let detections = [];
13
14 function setup() {
15 // cnv = createCanvas(500, 500);
16 cnv = createCanvas(windowWidth, windowHeight);
17
18 //Step 1: change the front camera to back camera
19 //video = createCapture(VIDEO);
20
21 //use phone's back camera
22 var constraints = {
23 audio: false,
24 video: {
25 facingMode: {
26 exact: "environment"
27 }
28 }
29 };
30 video = createCapture(constraints); //save the video results
31
32 // The line below + the videoLoadedCallback were added
33 // after the video was shot to fix compatibility issues.
34 video.elt.addEventListener('loadeddata', videoLoadedCallback);
35
36 // set video size
37 //video.size(500, 500);
38 video.size(windowWidth, windowHeight);
39 video.hide();
40
41
42 //Step 3: log data when people touch the screen
43 //cnv.touchEnded(logData);
44 }
45
46
47 function draw() {
48 //Step 2: draw video and detected objects
49 //draw video on canvas
50 image(video, 0, 0, video.width, video.height);
51
52 //draw bounding boxes for detected objects on canvas
53 for (let i = 0; i < detections.length; i++) {
54 let object = detections[i];
55 if(object.confidence > 0.6) {
56 stroke(0, 255, 0);
57 strokeWeight(4);
58 noFill();
59 rect(object.x, object.y, object.width, object.height);
}
}
}
}
```

Console Clear



# System Workflow





```

1  <!-- ego_network.json -->
2  {
3      "nodes": [
4          {"id": "person", "group": 1},
5          {"id": "bicycle", "group": 1},
6          {"id": "car", "group": 1},
7          {"id": "motorcycle", "group": 1},
8          {"id": "airplane", "group": 1},
9          {"id": "bus", "group": 1},
10         {"id": "train", "group": 1},
11         {"id": "truck", "group": 1},
12         {"id": "boat", "group": 1}
13     ],
14     "links": [
15         {"source": 0, "target": 1, "weight": 0.1},
16         {"source": 0, "target": 2, "weight": 0.3},
17         {"source": 0, "target": 3, "weight": 0.02},
18         {"source": 0, "target": 4, "weight": 0.01},
19         {"source": 0, "target": 5, "weight": 0.0},
20         {"source": 1, "target": 2, "weight": 0.5},
21         {"source": 1, "target": 3, "weight": 0.03},
22         {"source": 4, "target": 7, "weight": 0.6}
23     ]
24 }

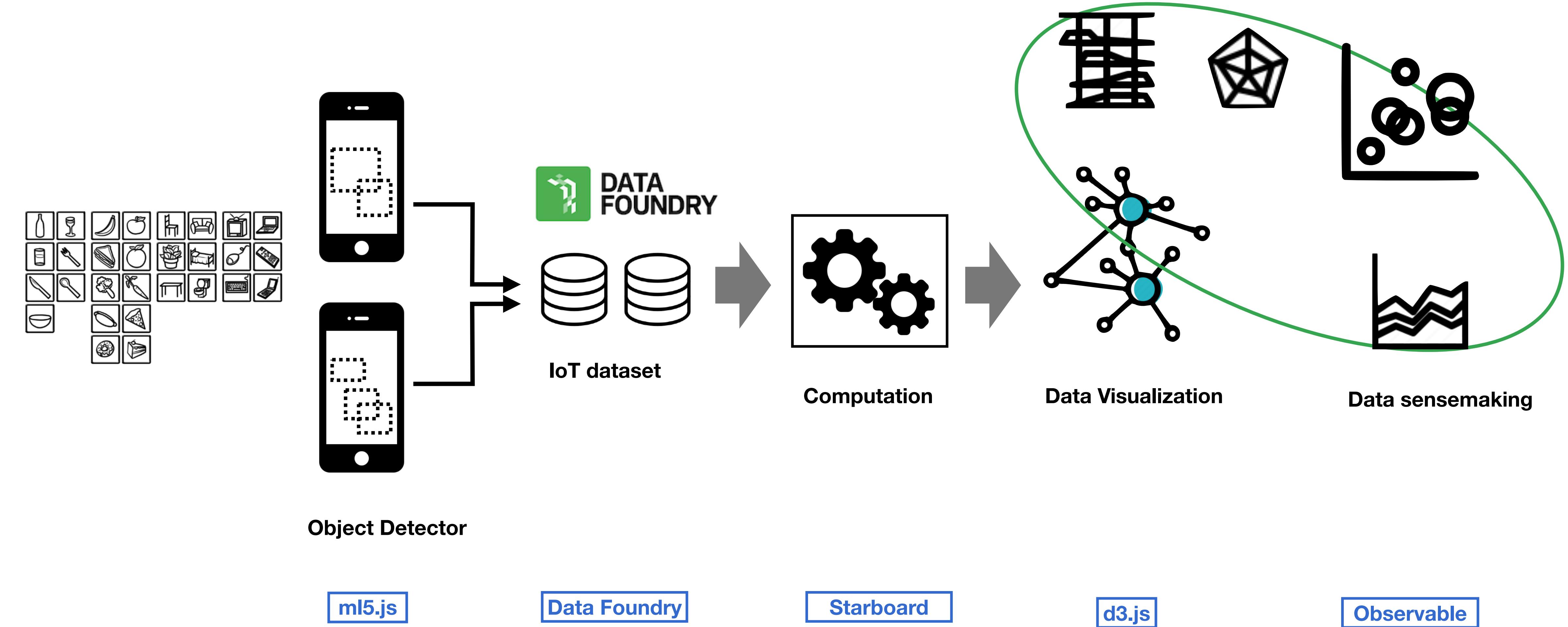
```

edit/create your own graph

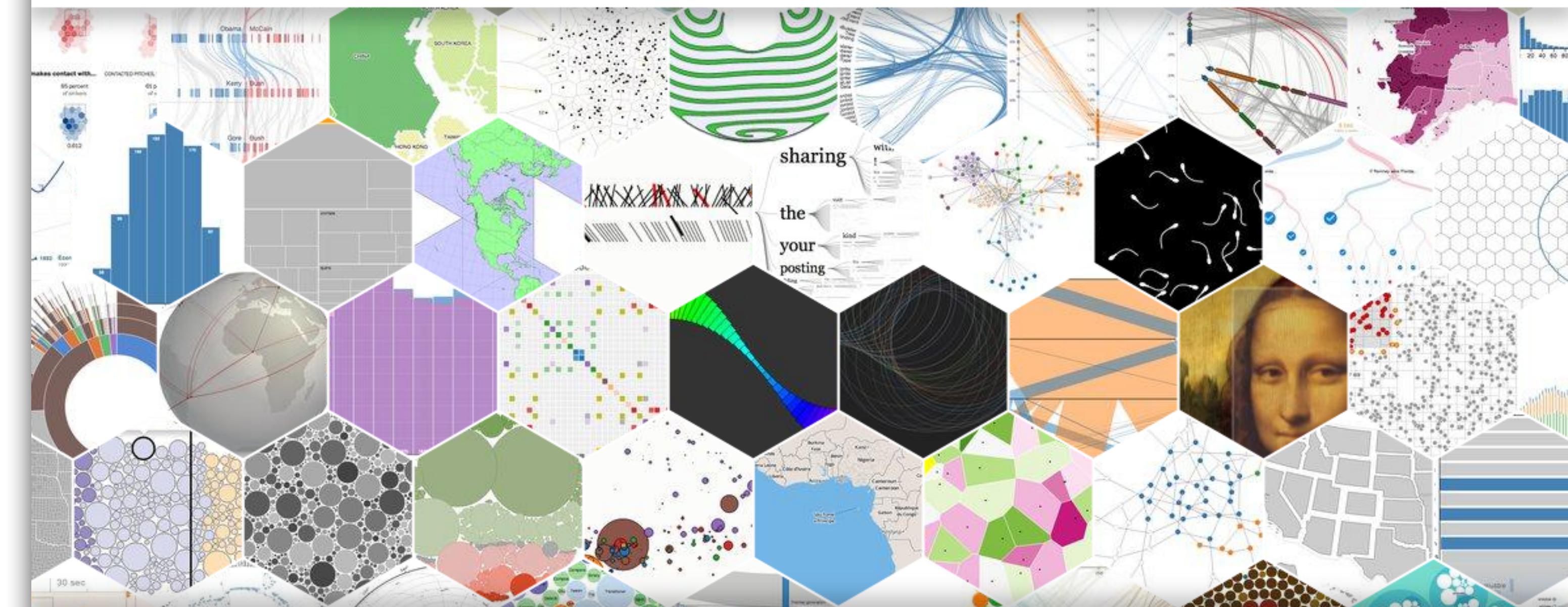


own\_graph.json

# System Workflow



# Data Visualization



Like visualization and creative coding? Try interactive JavaScript notebooks in [Observable!](#)

**D3.js** is a JavaScript library for manipulating documents based on data. **D3** helps you bring data to life using HTML, SVG, and CSS. D3's emphasis on web standards gives you the full capabilities of modern browsers without tying yourself to a proprietary framework, combining powerful visualization components and a data-driven approach to DOM manipulation.

Download the latest version (7.6.1) here:

- [d3-7.6.1.tgz](#)

To link directly to the latest release, copy this snippet:

```
<script src="https://d3js.org/d3.v7.min.js"></script>
```

The [full source and tests](#) are also available [for download](#) on GitHub.

- [See more examples](#)
- [Chat with the community](#)
- [Follow announcements](#)
- [Report a bug](#)
- [Ask for help](#)

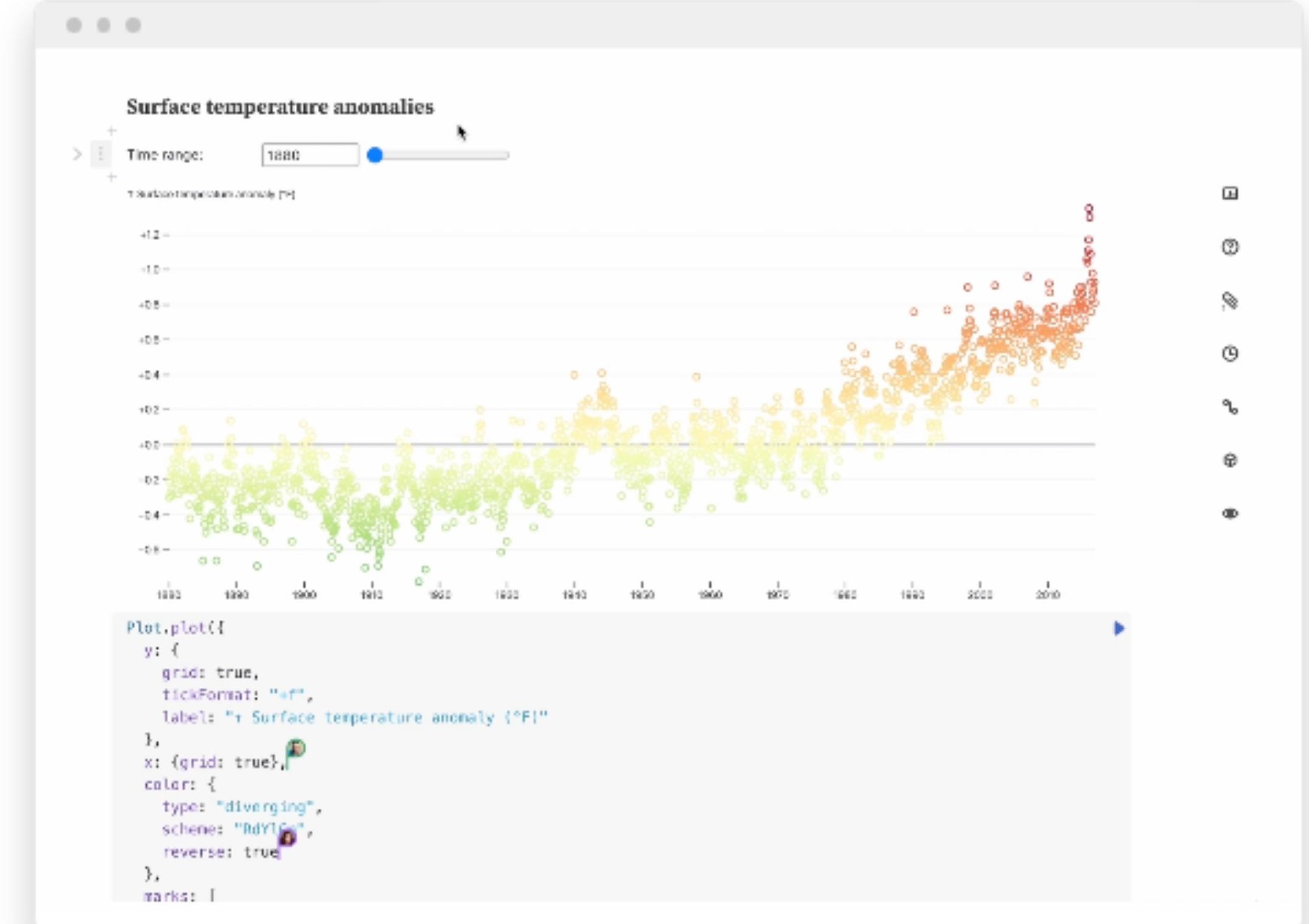
observablehq.com

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.



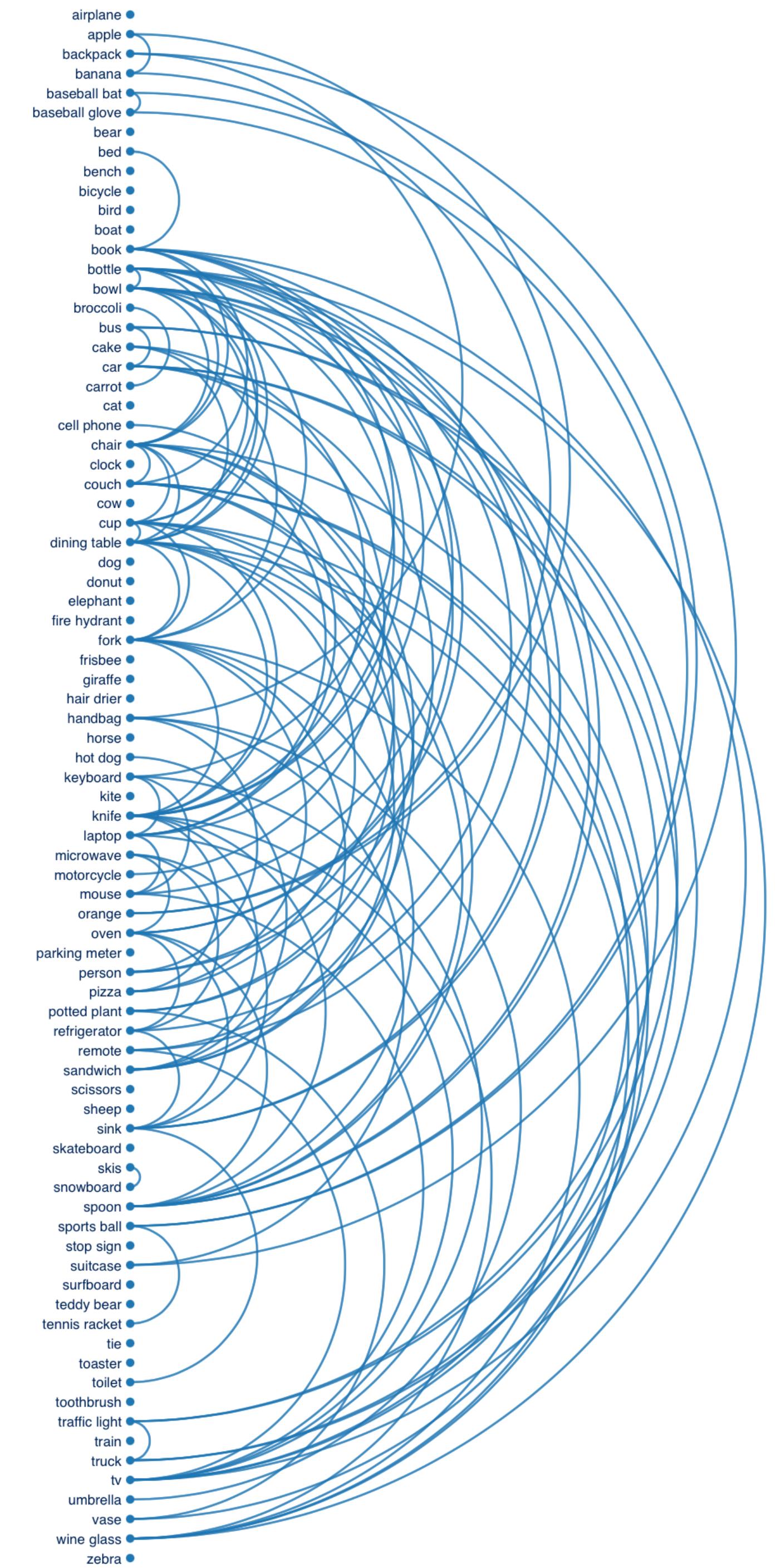
Uncover new insights, answer more questions, and make better decisions.

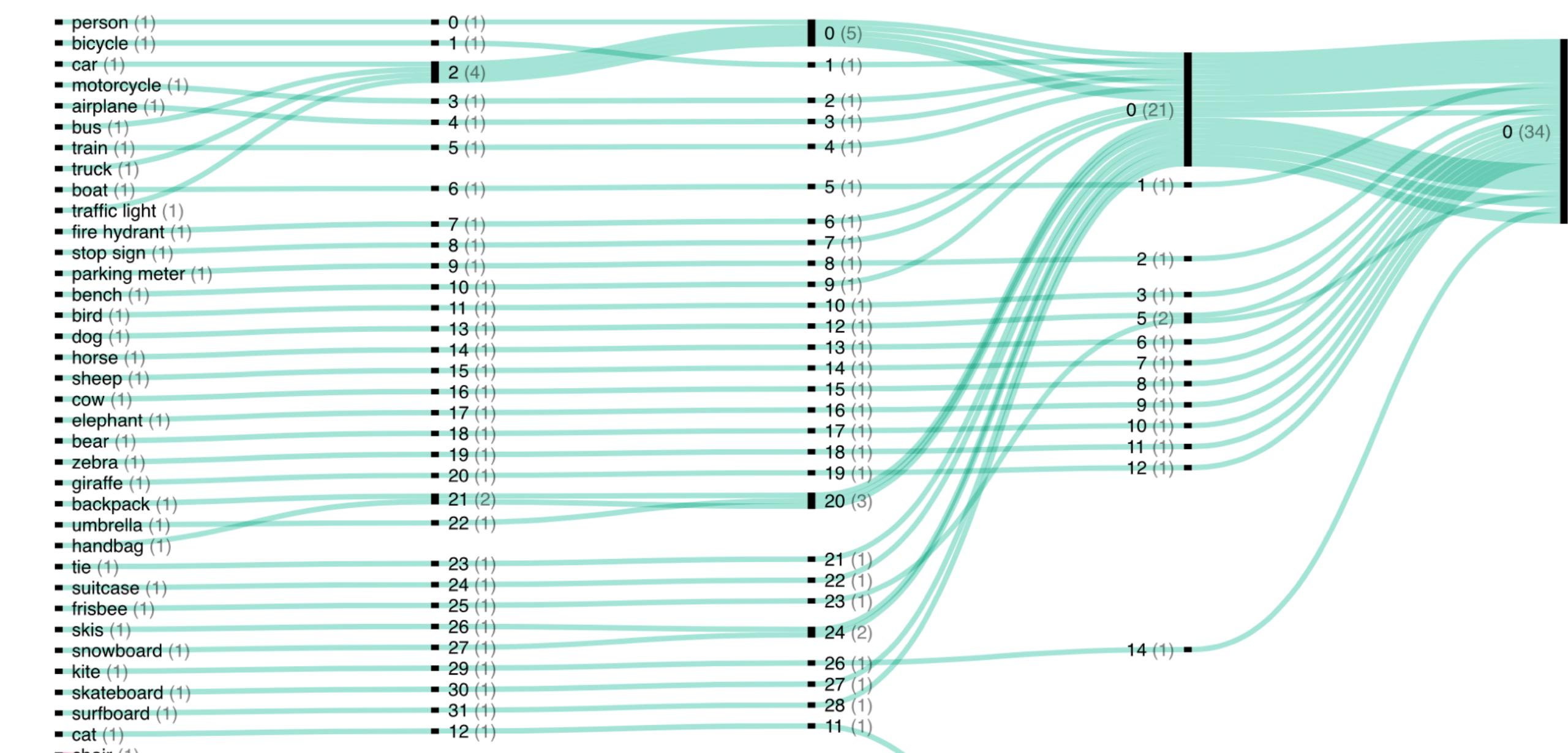
Sign up for free

<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.

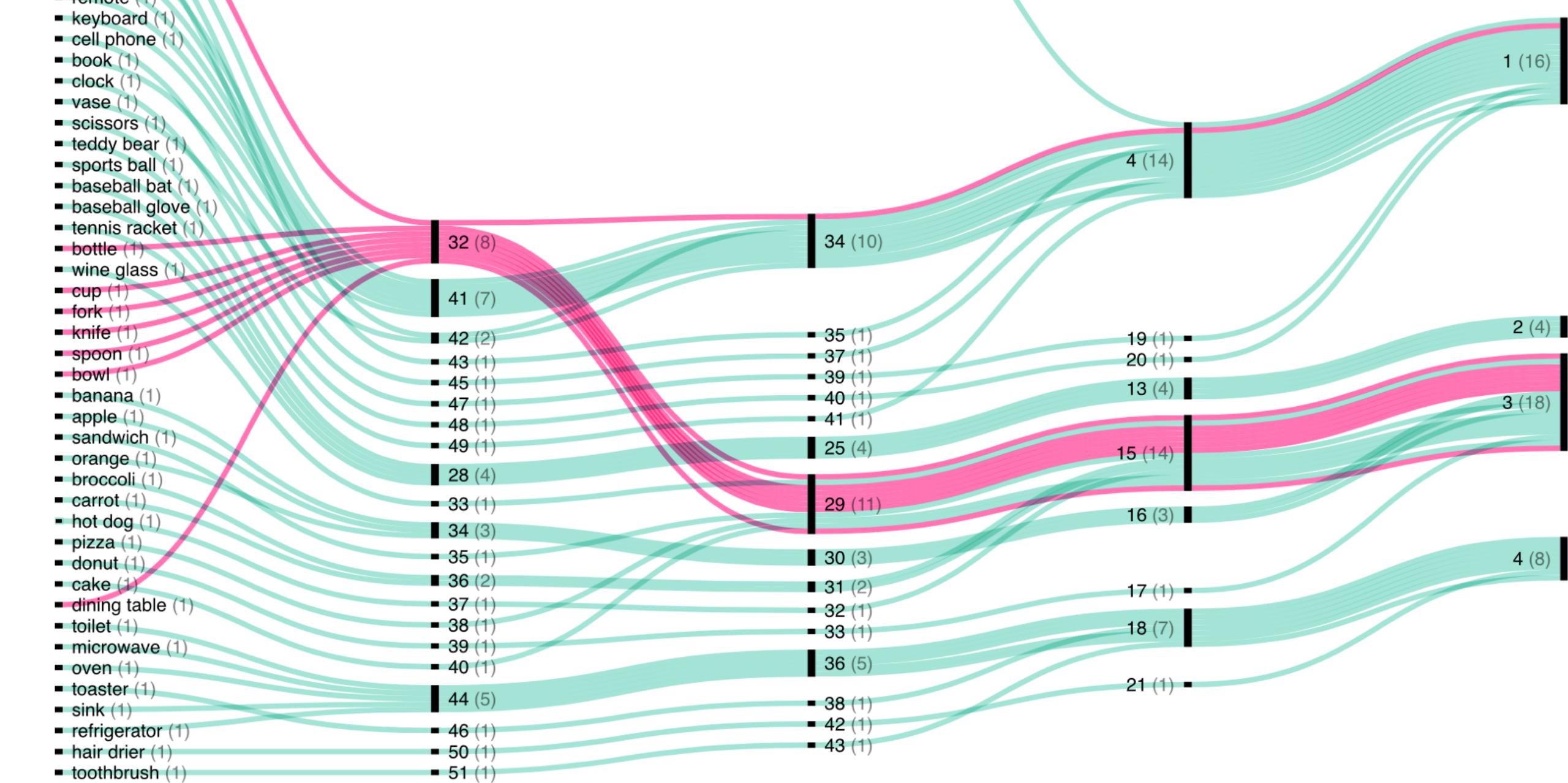




## Visualization: Parallel sets

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



## Visualization: Radial Tidy Tree

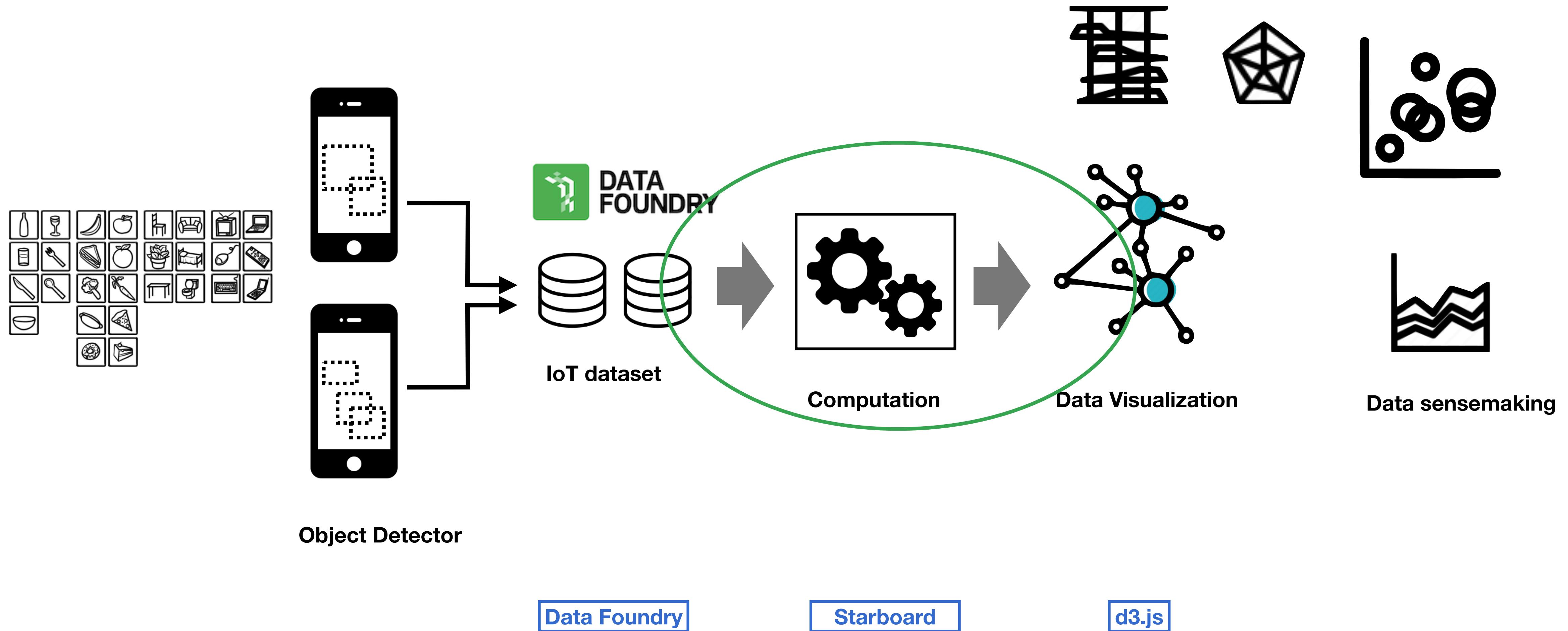
D3's [tree layout](#) implements the Reingold–Tilford “tidy” algorithm for constructing hierarchical node-link diagrams, improved to run in linear time by [Buchheim \*et al.\*](#). Tidy trees are typically more compact than [cluster dendograms](#), which place all leaves at the same level. See also the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



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

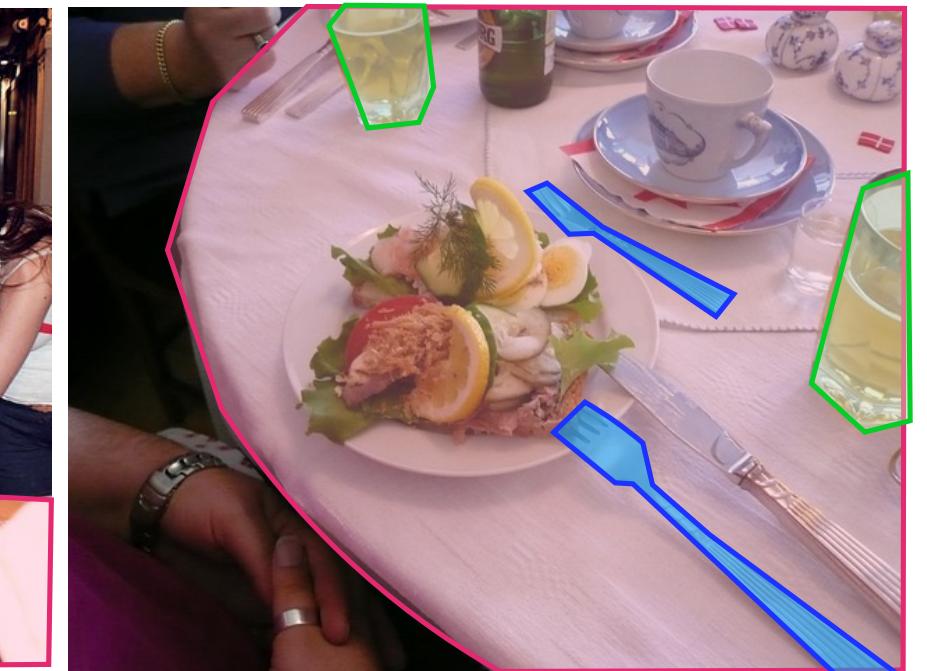
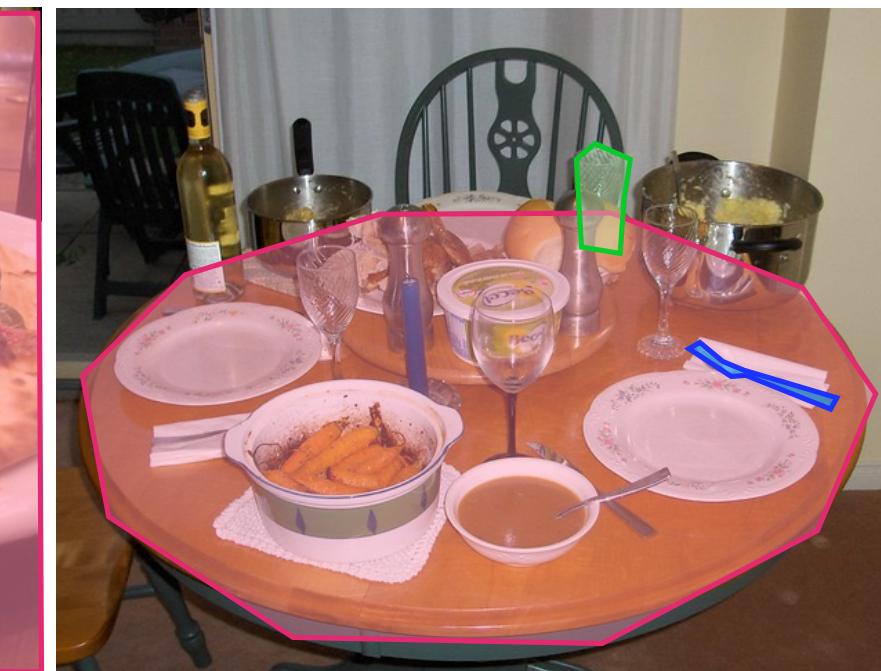
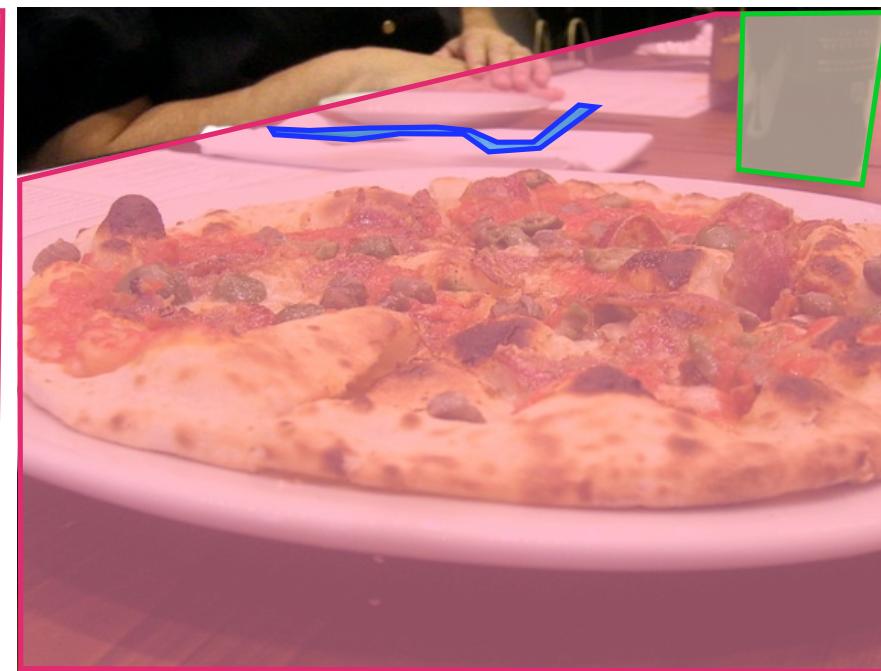
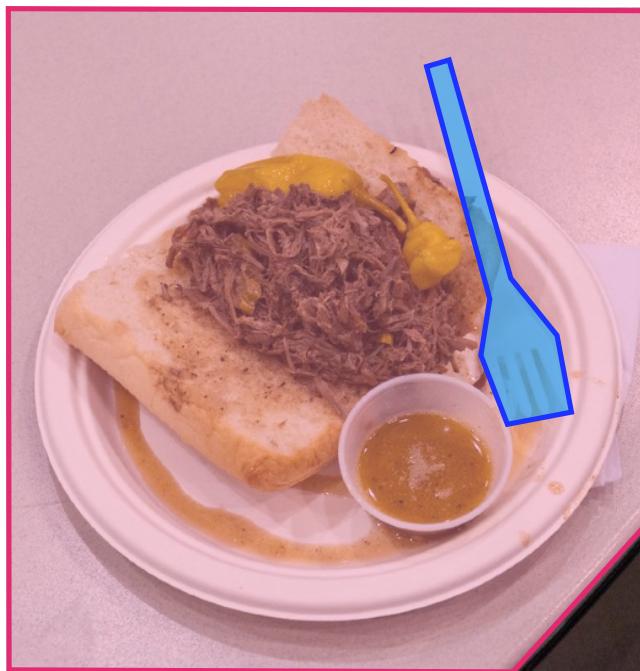
- (1) Read data from Data foundry**
- (2) Build object network based on collected data through similarity calculation (**starboard on DF**)**
- (3) Integrate all components (**hosting a web-based application on DF**)**

# System Workflow



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

-  cup
-  fork
-  dining table

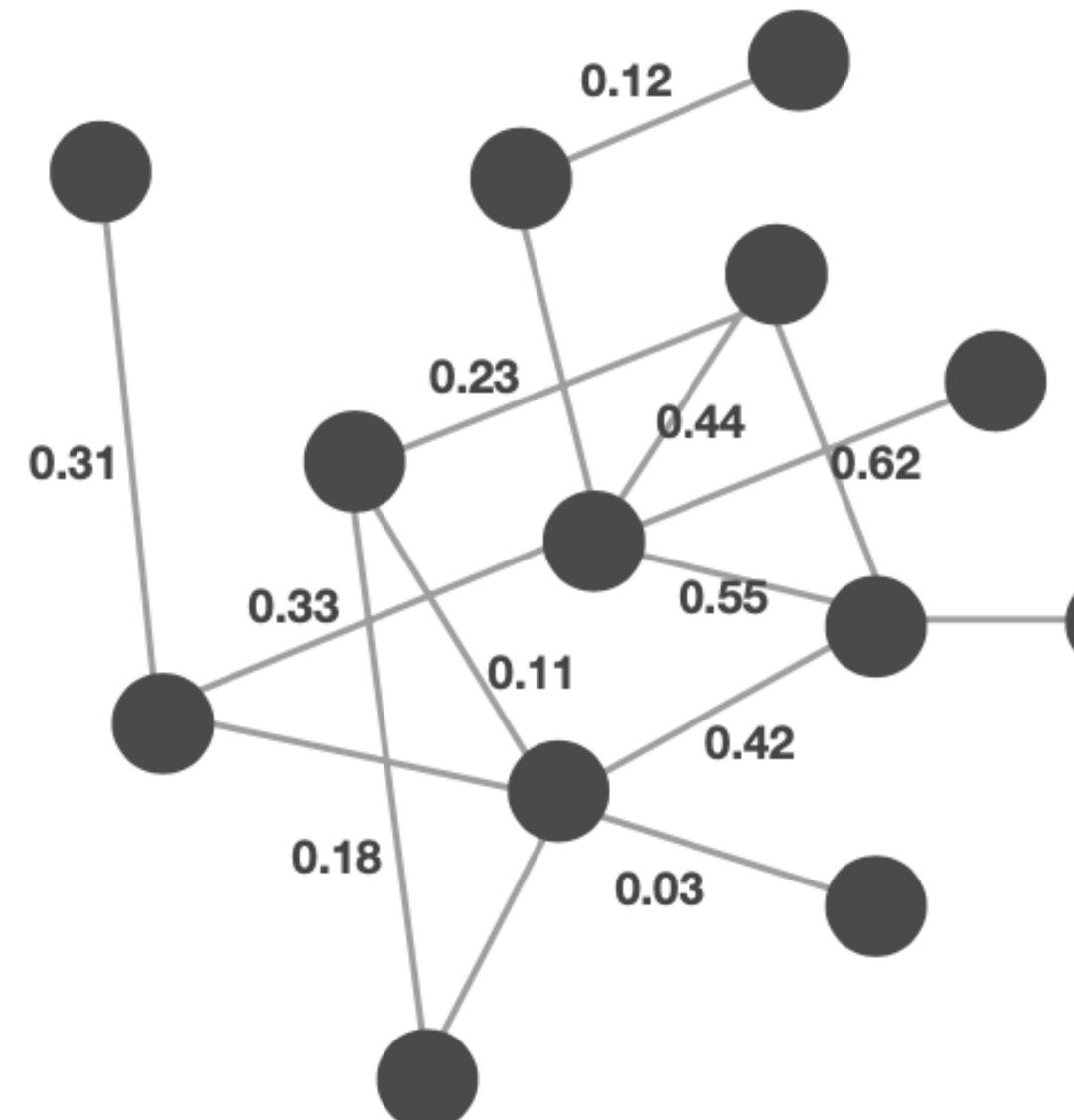


$$\text{Relation}(\text{Cup}, \text{Fork}) = \frac{|\text{Collection}_{\text{Cup}} \cap \text{Collection}_{\text{Fork}}|}{|\text{Collection}_{\text{Cup}} \cup \text{Collection}_{\text{Fork}}|}$$

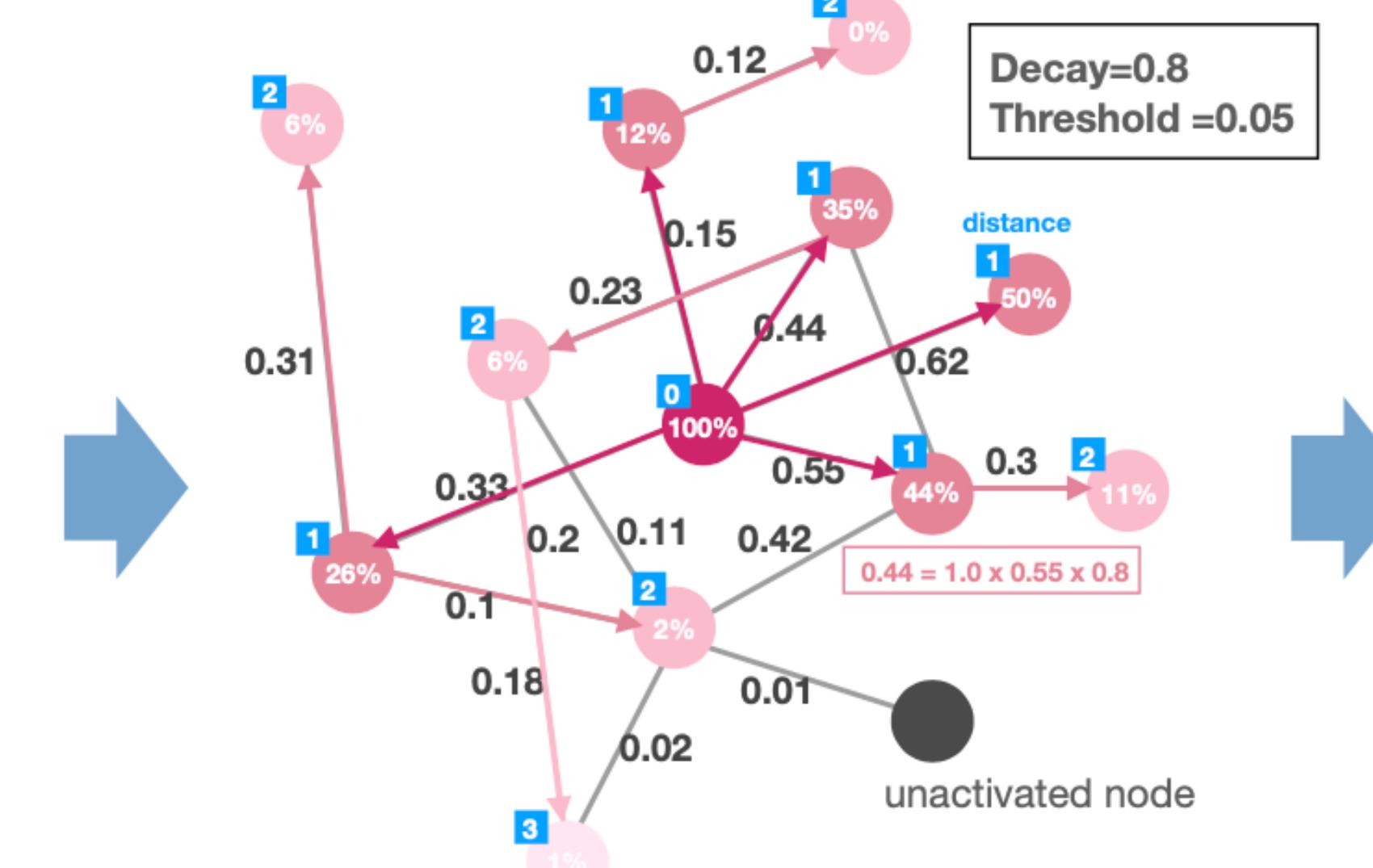
**4/5 = 0.80**

(Jaccard similarity coefficient)

# Construct Social-Centric and Ego-Centric Network

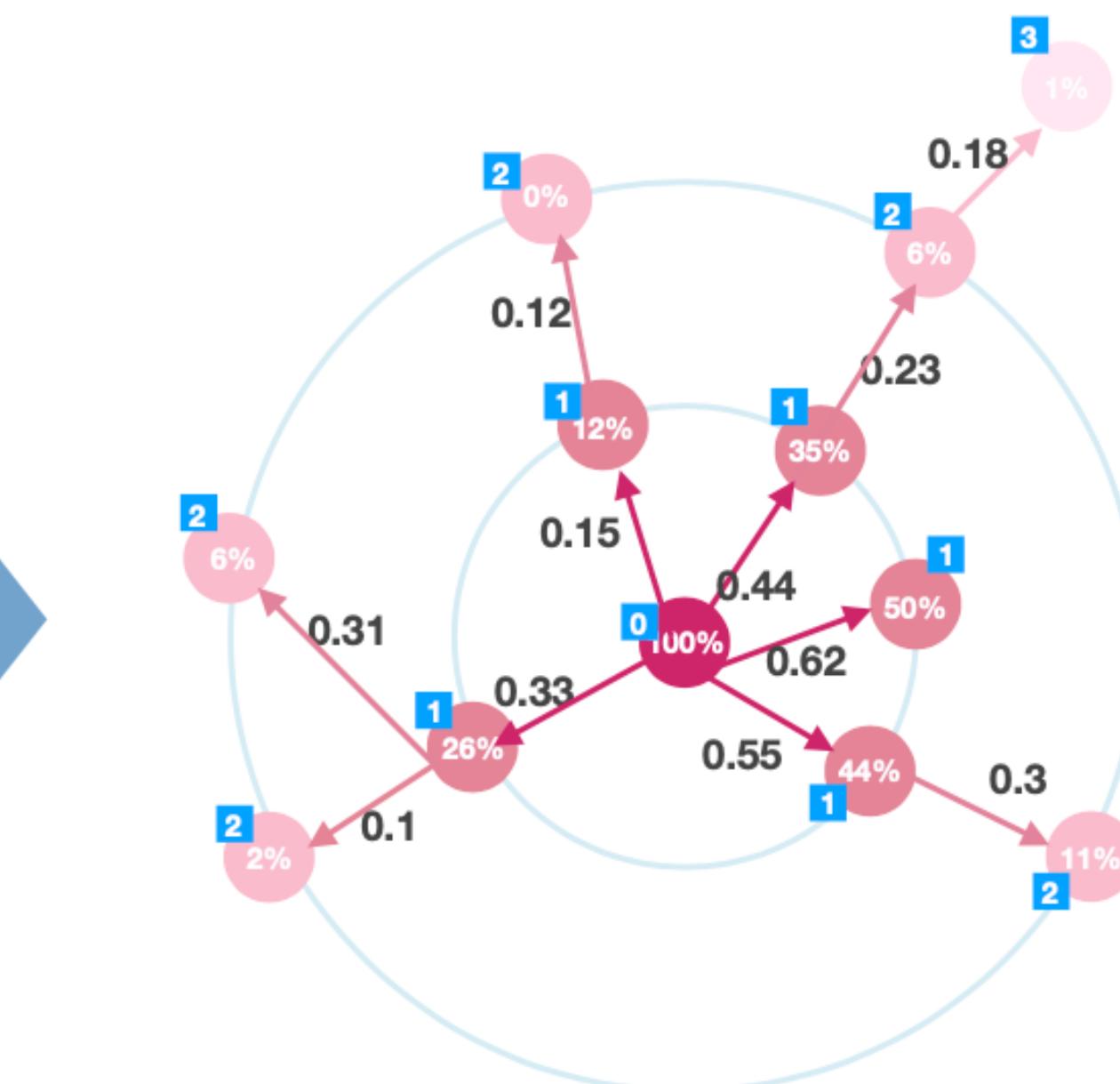


Social-Centric Network (undirected)



Spreading Activation

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



Ego-Centric Network (directed)



The shareable in-browser notebook

## 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](#).

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



[View source](#)

MARKDOWN

# 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, L<sup>A</sup>T<sub>E</sub>X, 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
```

# Use Starboard for interactive prototyping

The screenshot shows the Data Foundry interface. On the left, a sidebar menu includes options like Portfolio, My projects, Archive, Community, Collaborations, Subscriptions, Explore, Data tools, Guides, and Support. The main area displays a dataset titled "OBJECT DETECTOR USING ML5.JS". It contains sections for "Diary Dataset" (Data by participants as diary entries) and "Media Dataset" (Media files (images)). A pink box highlights the "Existing Dataset" section, which describes it as "One or more files of an existing dataset (data, text, images, audio, html)". Below this is a "Movement Dataset" section.

Step 1: Create an existing dataset

## Existing Dataset

One or more files of an existing dataset (data, text, images, audio, html)

## Movement Dataset

This screenshot shows the same Data Foundry interface as the previous one, but the main area now displays a dataset titled "[DCM210] AI WORKSHOP: CUSTOMIZED YOUR THINGCV". It includes a description of the workshop and its purpose. A pink box highlights the "Movement Dataset" section.

Step 2: Click to “Notebooks” button

This screenshot shows the Data Foundry interface with the "Notebooks" option selected in the sidebar. The main area lists datasets: "OBJECT DETECTOR USING ML5.JS", "OBJECT DATA", "THINGCV", and "WEB INTERFACE FOR CHATGPT". Each dataset entry includes an "EXISTING" status indicator and a "Web access" link. A pink box highlights the "Notebooks" button in the sidebar.

This screenshot shows the Data Foundry interface with the "Notebooks" option selected in the sidebar. The main area displays a list of notebooks. A pink box highlights the "ADD A NEW NOTEBOOK" button.

Home > Notebooks

## NOTEBOOKS

What are notebooks? Data Foundry notebooks is a web-based interactive computational environment for creating notebook documents.

A notebook document is a browser-based REPL containing an ordered list of input/output cells which can contain code, text (using Markdown), mathematics, plots and rich media. Underneath the interface, a notebook is a simple text document, usually ending with the ".gg" extension.

We are using the [Starboard system](#) for Data Foundry. In their own words: "Starboard brings cell-by-cell notebooks to the browser, no code is running the backend here! It's probably the quickest way to visualize some data with interactivity, do some prototyping, or build a rudimentary dashboard."

We have extended the Starboard interface with code examples and snippets to import your own datasets seamlessly. The best way to understand what notebooks can do in Data Foundry is to try one right away.

Notebook	Description	Uploaded
generate object_graph.gg		2022-09-15T15:10:21.089
generate object_graph.gg	object graph generator	2023-03-12T13:30:50.526
starBoardTest.gg (collaboration)		2021-10-26T11:36:00.688
copy.gg (collaboration)	staraobrd test2	2021-10-06T13:03:50.282

Step 3: Add a new notebook

This screenshot shows the Data Foundry interface with the "Notebooks" option selected in the sidebar. The main area displays a list of notebooks. A pink box highlights the "ADD A NEW NOTEBOOK" button.

This screenshot shows the Data Foundry interface with the "Notebooks" option selected in the sidebar. The main area displays a list of notebooks. A pink box highlights the "ADD A NEW NOTEBOOK" button.

Step 4: Add a notebook into the newly created existing dataset

## [DCM210] AI Workshop: Customized your thingCV --> Object detector

Test Object Detector --> Object detector

[DCM210] AI Workshop: Customized your thingCV --> Object detector using ml5.js

(using Markdown)

".gg" extension.

wser, no code is r

limentary dashba

est way to under

standing

the

best way to understand

the

Portfolio

My projects

Archive

Community

Collaborations

Subscriptions

Explore

Data tools

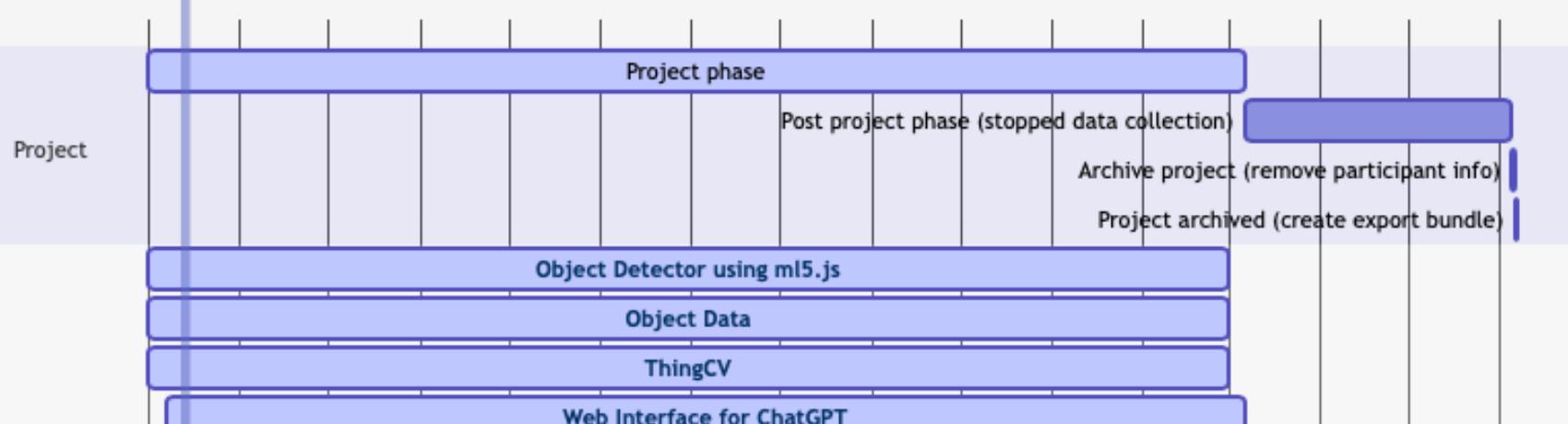
Guides

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 &gt; [DCM210] AI Workshop: Customized your thingCV &gt; Object graph generator

id: 7730 EXISTING PUBLIC MIT

## OBJECT GRAPH GENERATOR

2023-03-01

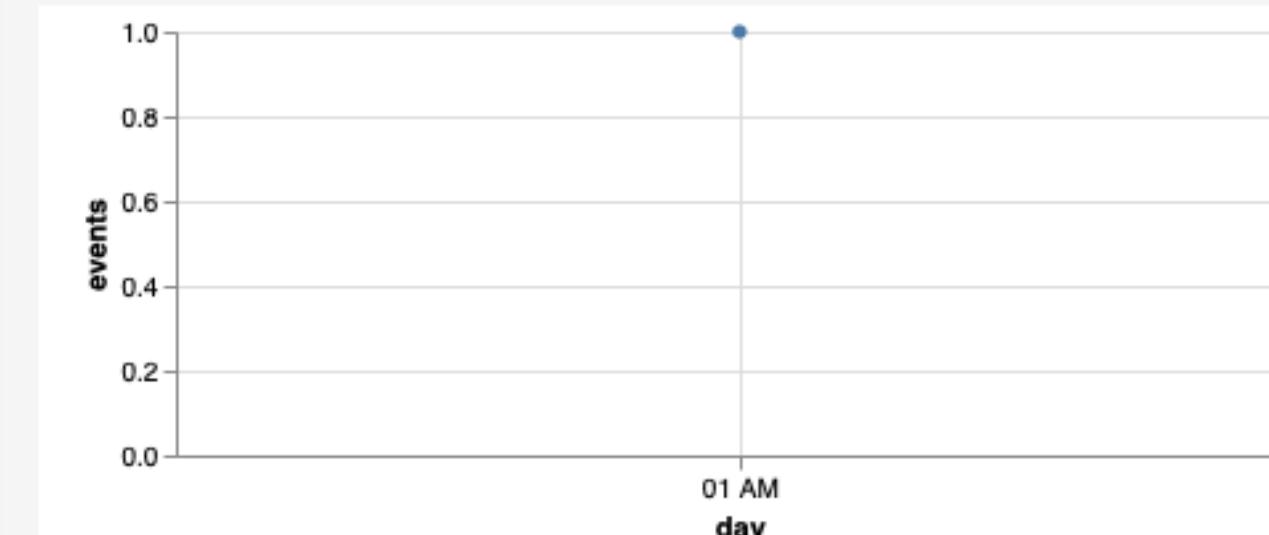
2024-03-01

generate object graph based on data collected from DF

INFO



License: MIT



### DATASET FILES

File name	Description	Uploaded	Actions
generate object_graph.gg	object graph generator	Mar 12 at 13:30	<a href="#">edit</a>   <a href="#">delete</a>

VIEW DATA

DOWNLOAD

UPLOAD FILE(S)

[<< 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](#)

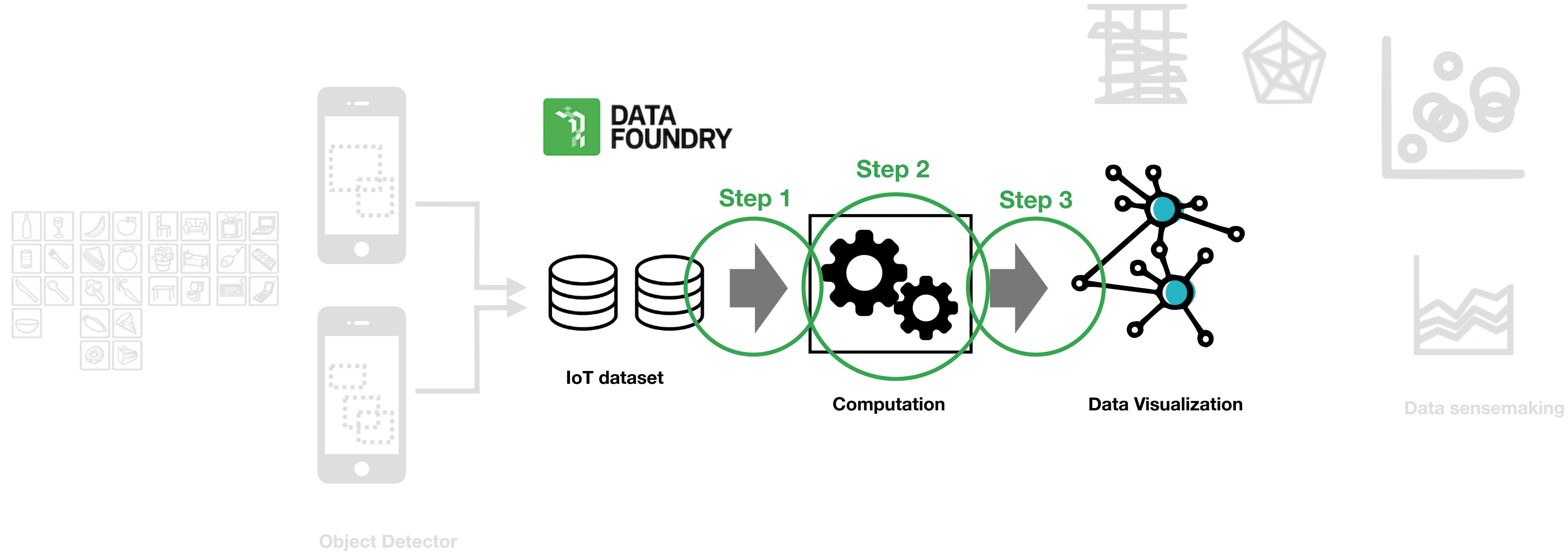
---

JAVASCRIPT...

```
1 let what = {  
2     test: true  
3 }  
4  
5 console.log(what)
```

PYTHON...

```
1 globalWhat = {123}  
2  
3 print(globalWhat)  
4
```



- (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.ggb

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.

[https://data.id.tue.nl/datasets/existing/edit/7730/1/generate%20object\\_graph.ggb](https://data.id.tue.nl/datasets/existing/edit/7730/1/generate%20object_graph.ggb)

Save notebook

## 1. Read data from Data foundry

```
//read data from DF
1 data_url = "https://data.id.tue.nl/datasets/downloadPublic/json/M1pxNVIrWU8zMUg4L21zN0JwbU1WNjBxUmVLa0ZiRnVwVlc5L2
2 var jdata = await fetch(data_url);
3 var jsonData = await jdata.json();
4
5 var object_data = [];
6 for(d in jsonData){
7   if(jsonData[d].detections != ""){
8     object_data.push(jsonData[d].detections.split(","));
9   }
10  }
11 }
12 console.log(object_data);

(29) [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)

  1 obj_img_map = {}
  2 object_data.forEach((obj_list, indx) => {
  3   obj_list.forEach(obj => {
  4     if(obj in obj_img_map){
  5       obj_img_map[obj].push(indx)
  6     }else{
  7       obj_img_map[obj] = []
  8       obj_img_map[obj].push(indx)
  9     }
10   })
11 }
12 }
13 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

```
1 object_net = {}
2
3 for(obj_1 in obj_img_map){
4   if(!(obj_1 in object_net)){
5     object_net[obj_1] = {};
6   }
7 }
8
9 for(obj_2 in obj_img_map){
10  if(obj_2 == obj_1){
11    object_net[obj_1][obj_2] = 1.0;
12    continue;
13  }
14
15  if(!(obj_2 in object_net[obj_1])){
16    let intersection = obj_img_map[obj_1].filter(x => obj_img_map[obj_2].includes(x));
17    let union = [...obj_img_map[obj_1], ...obj_img_map[obj_2]];
18
19    score = (new Set(intersection).size*1.0 / (new Set(union)).size);
20  }
21  object_net[obj_1][obj_2] = score;
22 }
23 }
24
25 for(obj in object_net){
26   console.log(object_net[obj]);
27 }
```

```
  ▶ {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}
```

## 3. Calculate similarity score for each link

## Notebook: generate object\_graph.ggb

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](#)

```
//read data from DF
data_url = "https://data.id.tue.nl/datasets/downloadPublic/json/M1pxNViTmU8zMtg4L21zN0JwbU1WNjBxUmVL08ZiRnVwV1cSLI"
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);
```

```
//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"]
]
```

```
obj_img_map = {}
data.forEach((obj_list, idx) => {
    obj_list.forEach(obj => {
        if(obj in obj_img_map){
            obj_img_map[obj].push(idx)
        }else{
            obj_img_map[obj] = []
            obj_img_map[obj].push(idx)
        }
    })
})
console.log(obj_img_map);
```

```
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 = [...new Set(obj_img_map[obj_1]), ...new Set(obj_img_map[obj_2])];
            let jaccard = intersection.length / union.length;
            object_net[obj_1][obj_2] = jaccard;
        }
    }
}
```

## 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.

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

217 var root = {};

218 var readDataFromDF = async function() {
219     data_url = "https://data.id.tue.nl/datasets/downloadPublic/json/M1pxNViTmU8zMtg4L21zN0JwbU1WNjBxUmVL08ZiRnVwV1cSLI";
220     var jdata = await fetch(data_url);
221
222     var jsonData = await jdata.json();
223     var object_data = [];

224     for(d in jsonData){
225         if(jsonData[d].detections != ""){
226             object_data.push(jsonData[d].detections.split(","));
227         }
228     }
229
230     graph = calculateGraph(object_data);
231     return graph;
232 }
233
234 function calculateGraph(data){
235     obj_img_map = {}
236     data.forEach((obj_list, idx) => {
237         obj_list.forEach(obj => {
238             if(obj in obj_img_map){
239                 obj_img_map[obj].push(idx)
240             }else{
241                 obj_img_map[obj] = []
242                 obj_img_map[obj].push(idx)
243             }
244         })
245     })
246
247     object_net = {}
248     for(obj_1 in obj_img_map){
249         if(!(obj_1 in object_net)){
250             object_net[obj_1] = {};
251
252             for(obj_2 in obj_img_map){
253                 if(obj_2 == obj_1){
254                     object_net[obj_1][obj_2] = 1.0;
255                     continue;
256                 }
257
258                 if(!(obj_2 in object_net[obj_1])){
259                     let intersection = obj_img_map[obj_1].filter(x => obj_img_map[obj_2].includes(x));
260                     let union = [...new Set(obj_img_map[obj_1]), ...new Set(obj_img_map[obj_2])];
261                     let jaccard = intersection.length / union.length;
262                     object_net[obj_1][obj_2] = jaccard;
263                 }
264             }
265         }
266     }
267 }
```

SAVE

CANCEL

# **Session II: Build your LLMs model on DF**

**GET STARTED**[Introduction](#)[Quickstart](#)[Libraries](#)[Models](#)[Tutorials](#)[Data usage policies](#)[Usage policies](#)**GUIDES**[Text completion](#)[Code completion](#)[Chat completion](#)[Introduction](#)[Instructing chat models](#)[Chat vs Completions](#)[FAQ](#)[Image generation](#)[Fine-tuning](#)[Embeddings](#)[Speech to text](#)[Moderation](#)[Rate limits](#)[Error codes](#)[Safety best practices](#)[Production best practices](#)**API REFERENCE**[Introduction](#)[Authentication](#)[Making requests](#)[Models](#)[Completions](#)[Chat](#)[Edits](#)[Images](#)[Embeddings](#)

## 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:

```
1 # Note: you need to be using OpenAI Python v0.27.0 for the code below to run
2 import openai
3
4 openai.ChatCompletion.create(
5     model="gpt-3.5-turbo",
6     messages=[
7         {"role": "system", "content": "You are a helpful assistant."},
8         {"role": "user", "content": "Who won the world series in 2020?"}
9         {"role": "assistant", "content": "The Los Angeles Dodgers won the World Series in 2020."}
10        {"role": "user", "content": "Where was it played?"}
11    ]
12 )
```

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.

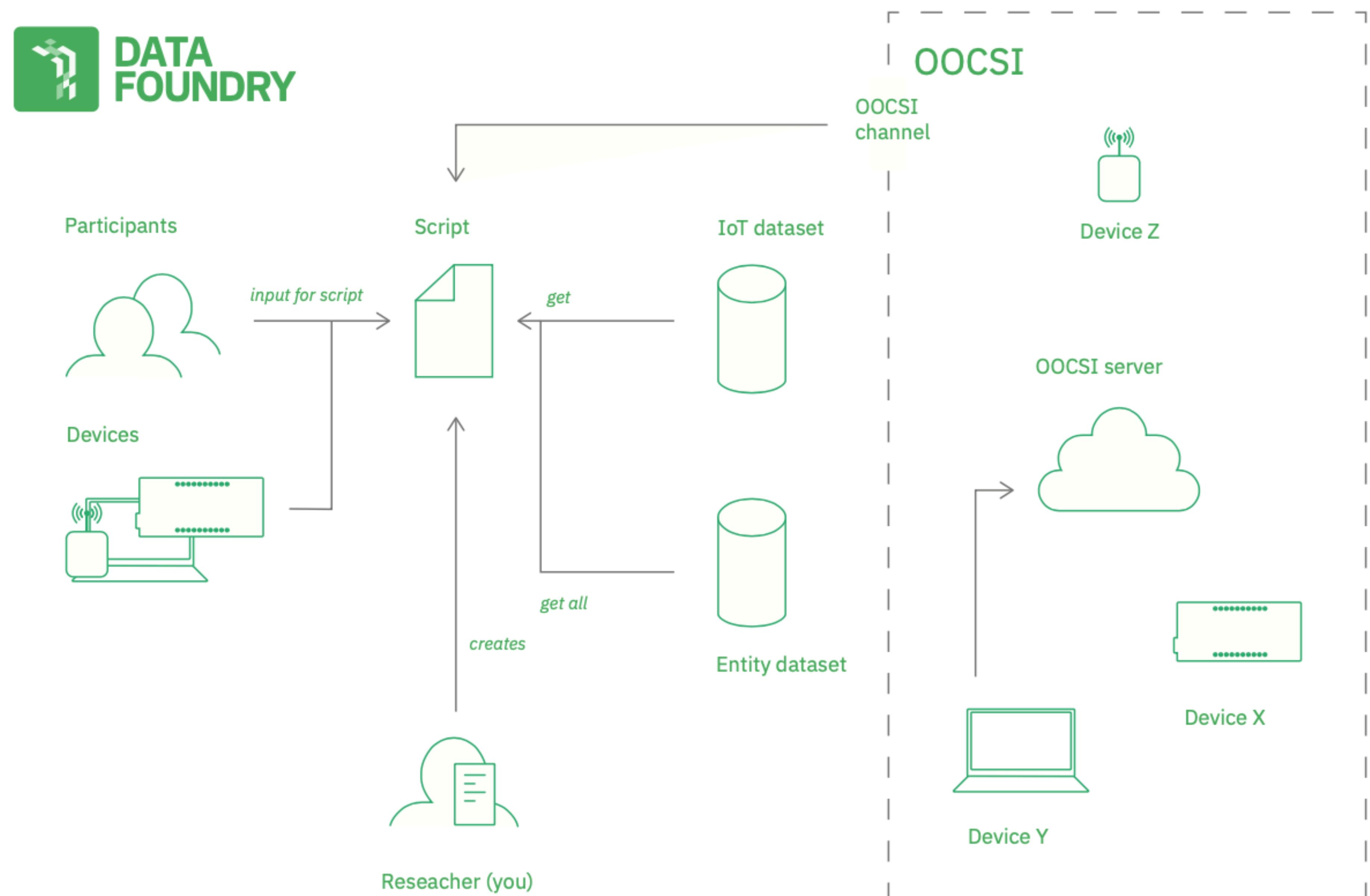
## Response format

An example API response looks as follows:

<https://platform.openai.com/docs/guides/chat>

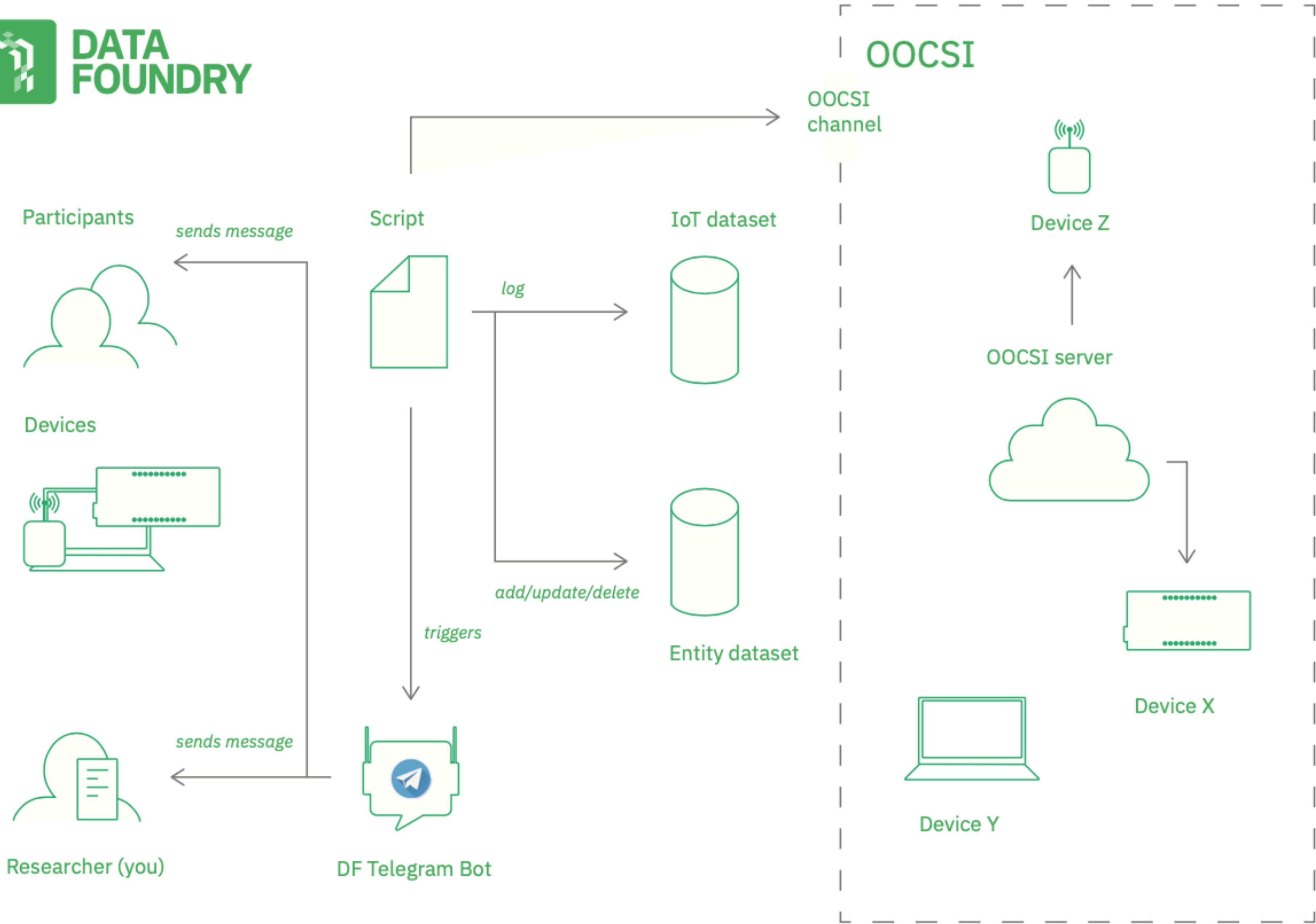
<https://platform.openai.com/docs/api-reference/chat>

## Script Input





## Script Output





Welcome to OOCSI

**What it is**

A prototyping middleware for design education and research designed and developed at the Industrial Design department of Eindhoven University of Technology. OOCSI is completely open-source and published under a permissive license. Consider contributing on [GitHub](#).

**How to learn more?**

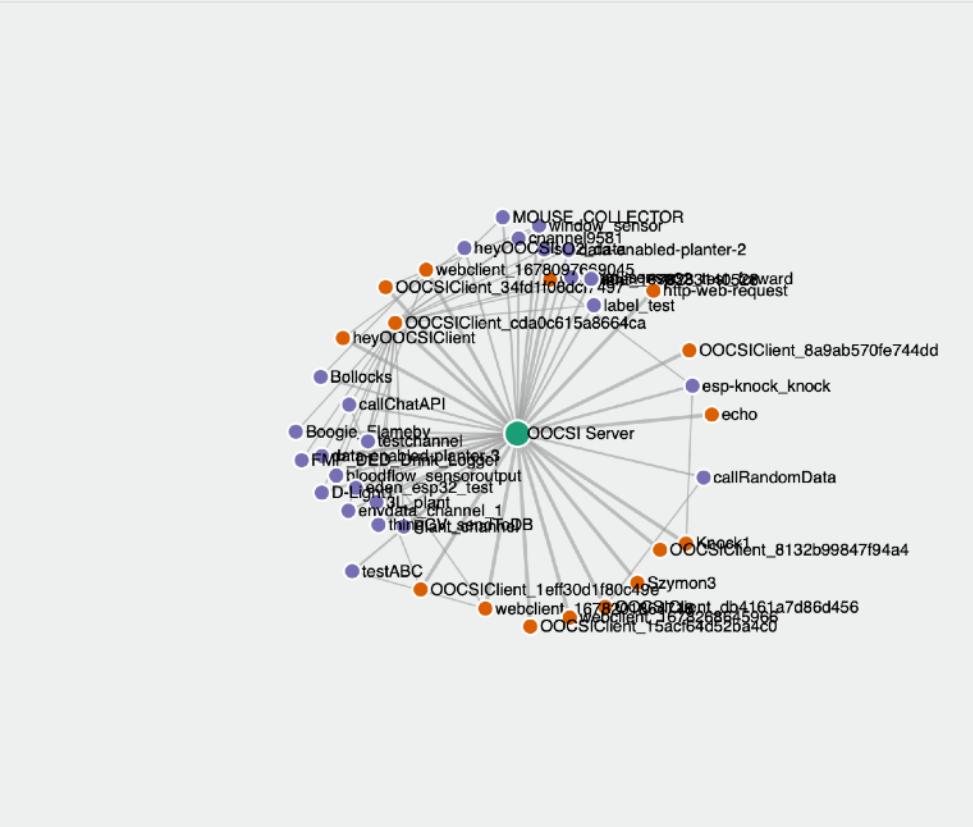
We are currently preparing some more information material about OOCSI and a series of short video lectures. In the meanwhile, you can read more about OOCSI on the [wiki](#).

**What is going on right now**

An OOCSI server can be busy at times; currently, this is happening:  
Clients: 17, Channels: 43, Messages: 172809, Messages/sec: 2  
More [metrics](#).

The following clients are currently connected (refresh page to update): [Knock1](#), [heyOOCSTClient](#), [http-web-request](#), [OOCSTClient\\_db4161a7d86d456](#), [Szymon3](#), [echo](#), [OOCSTClient\\_1eff30d180c49e](#), [webclient\\_1678283140528](#), [webclient\\_1678201864718](#), [webc...](#)

The following channels are currently open (refresh page to update): [heyOOCSTClient](#), [data-enabled-planter-2](#), [OOCSTClient\\_db4161a7d86d456](#), [data-enabled-planter-3](#), [echo](#), [3L\\_planet](#), [callChatAPI](#), [callRandomData](#), [bloodflow\\_sensoroutput](#), [webclie...](#)



With these different languages OOCSI can be accessed on most popular platforms on desktop (Windows, Mac, Linux), mobile (iOS, Android) and embedded (Raspberry Pi, ESP, ...).

[bigger version](#)

(Your client does not have any emoji? Well, check [heyOOCSTClient](#) how to add one.)

**IMPORTANT NOTE**

This is a public server that allows open connectivity. Please use it responsibly and be aware that all data is in principle visible to all other users. Do NOT send confidential, personal or secret information. Fair use applies.

The server will not store any information. Log files may contain OOCSI-specific connection meta-data, but no personally identifiable information such as IP addresses, email addresses etc.

**What you can do here (on this server)**

- Build and test prototypes that need connectivity
- Test and evaluate OOCSI for your use-case
- Try some examples

How to do that? Check out the connection options below.

**How to connect**

OOCSI can be used by many different platforms and has libraries or bindings for different programming languages. See below for some options:

- Processing
- Arduino / ESP (C/C++)
- Python/Micropython
- Websocket (Javascript)

Connect to this server:

```
OOCSI.connect('wss://oocsi.id.tue.nl/ws')
```

More information: [setup & use](#).

- Java/Android
- Command line

# OOCSI

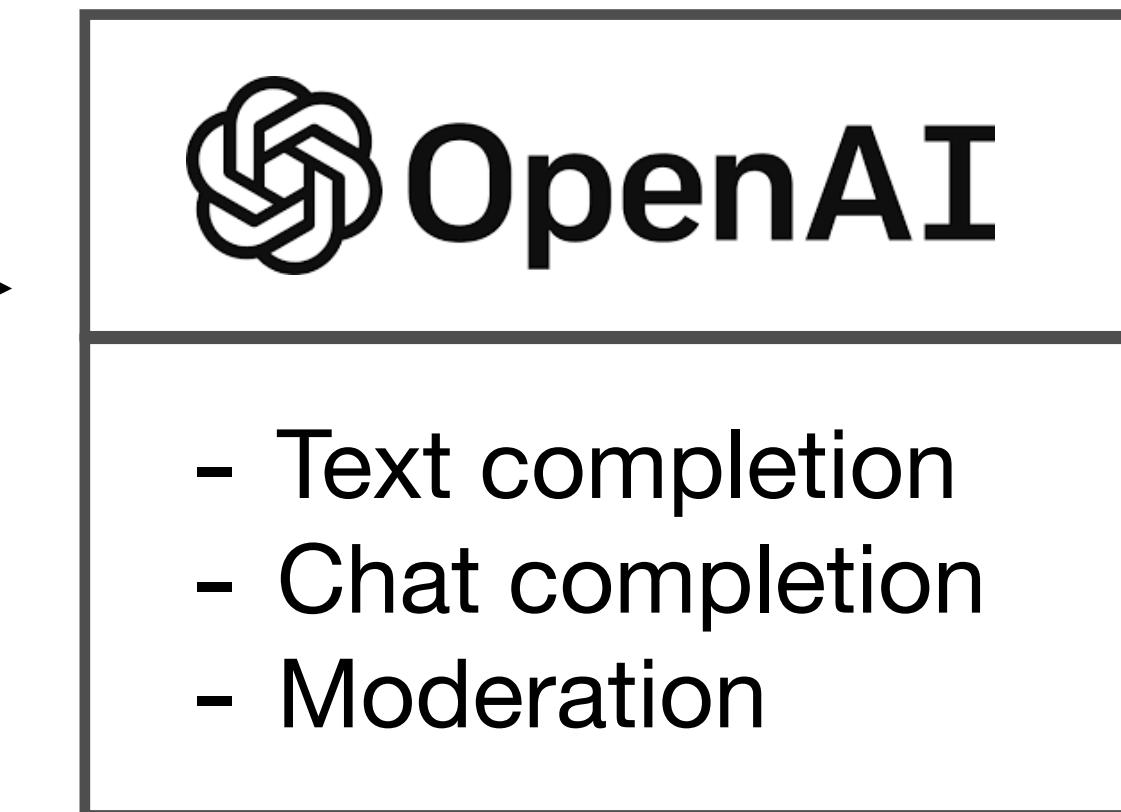
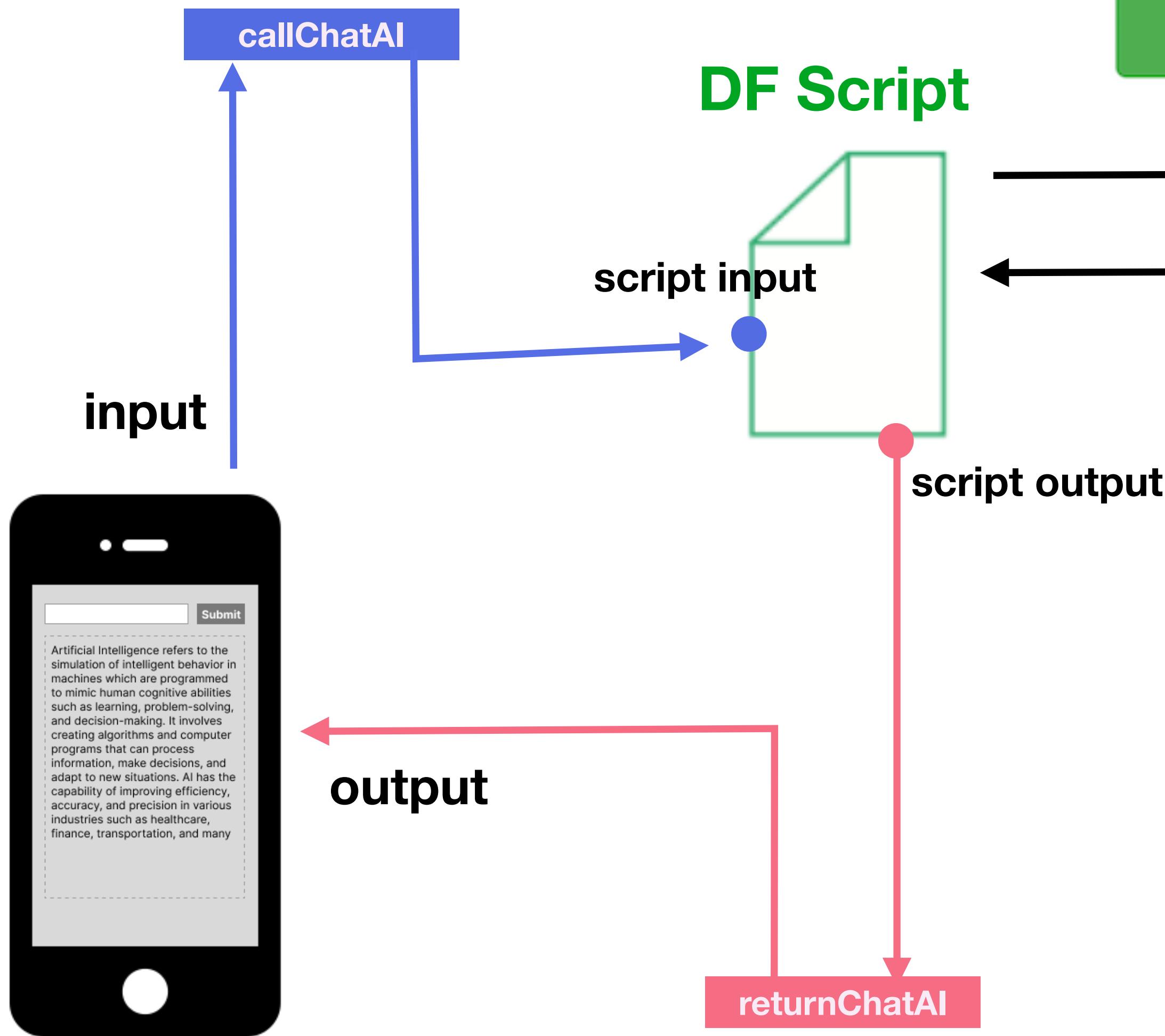
OOCSI is a prototyping “middleware” for designing distributed products and it is targeted mainly at Industrial Design and Computer Science education.

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/>

**OOCSI**



## 4 Models on DF:

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



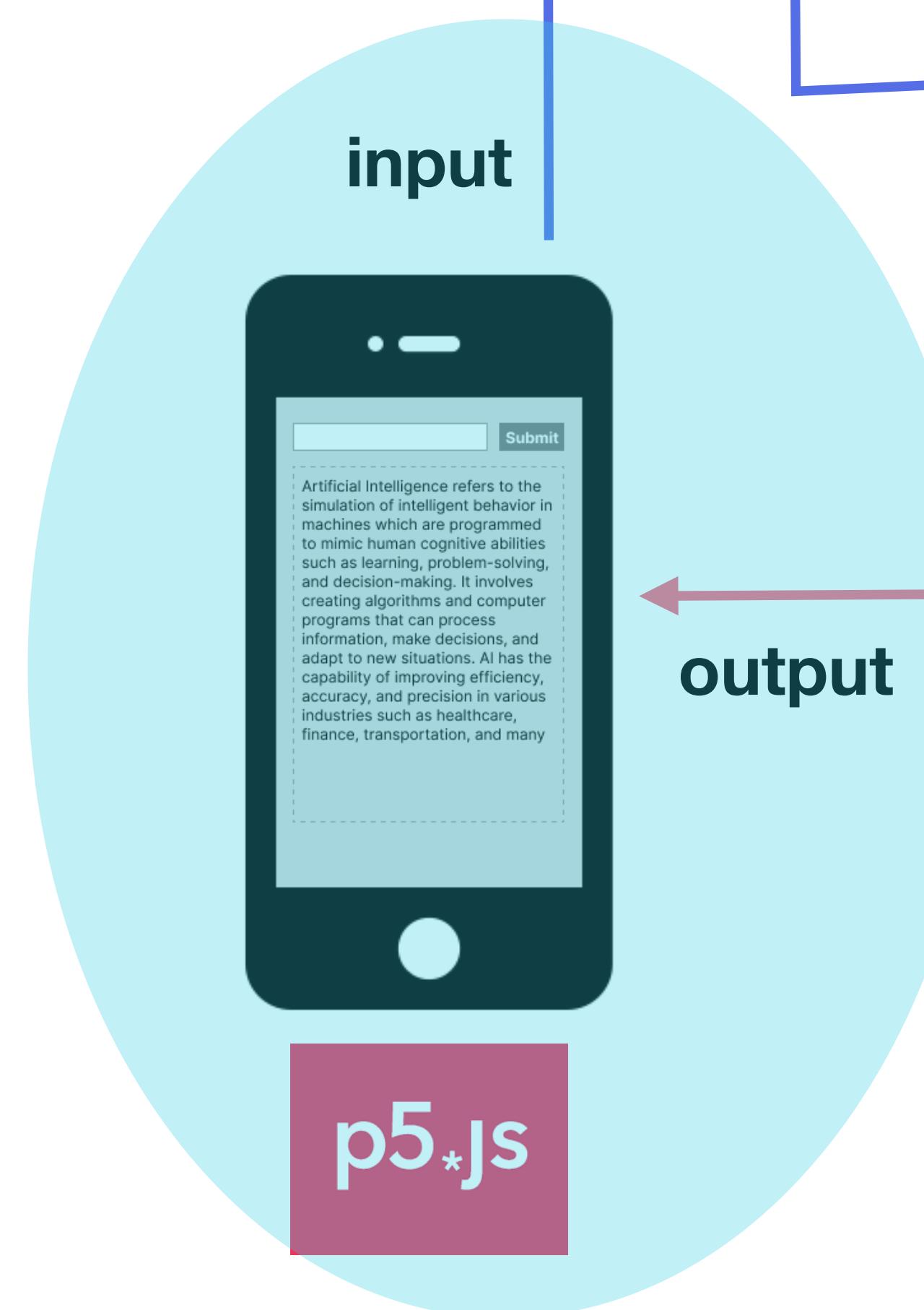
**OOCSI**

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



**OOCSI**

**callChatAI**



**DF Script**

**script input**

**script output**



**OpenAI**

- Text completion
- Chat completion
- Moderation

## 4 Models on DF:

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



**OOCSI**

 Auto-refresh

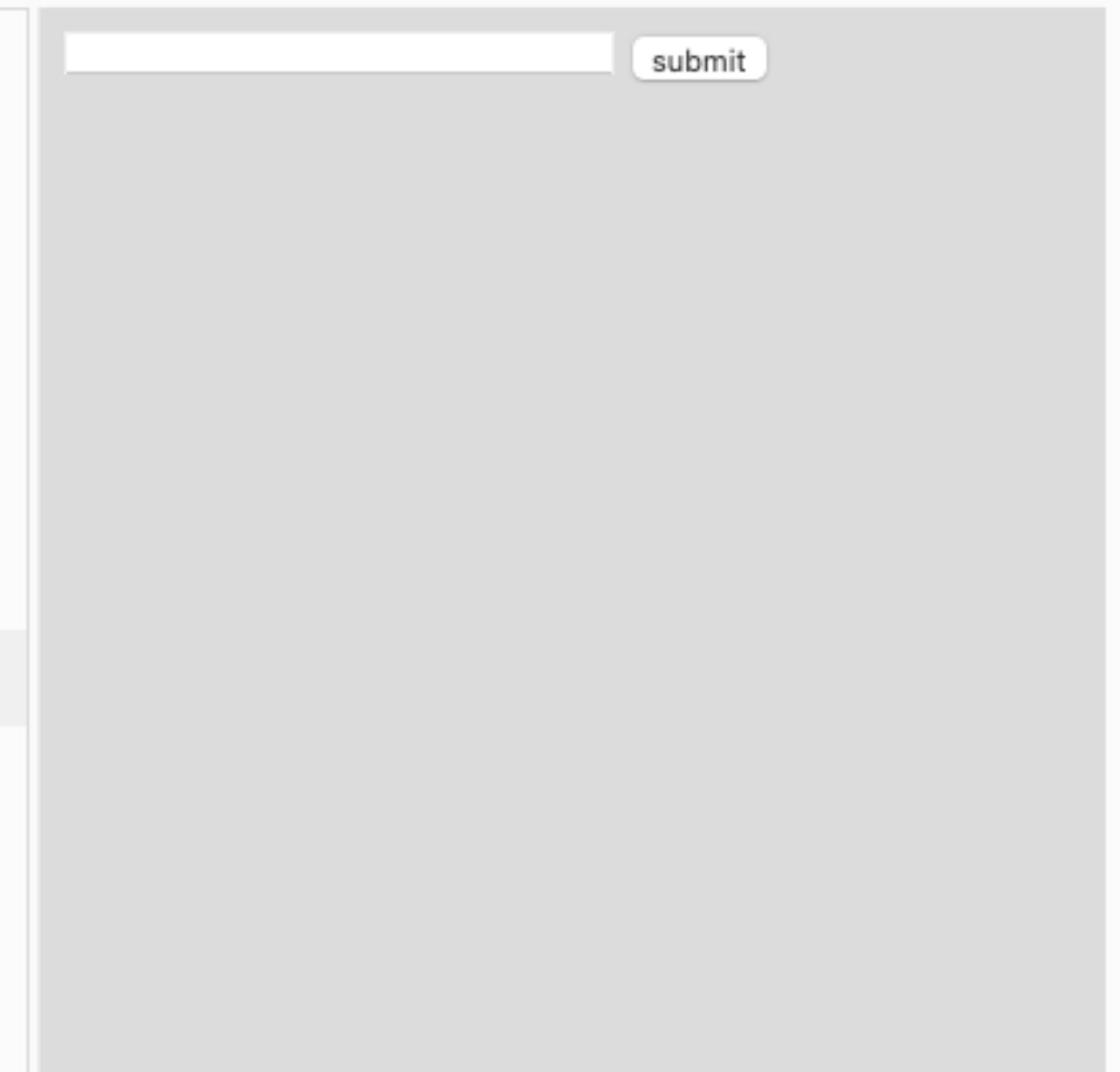
Test OpenAPI on DF by janetyc

&gt; sketch.js

Saved: just now

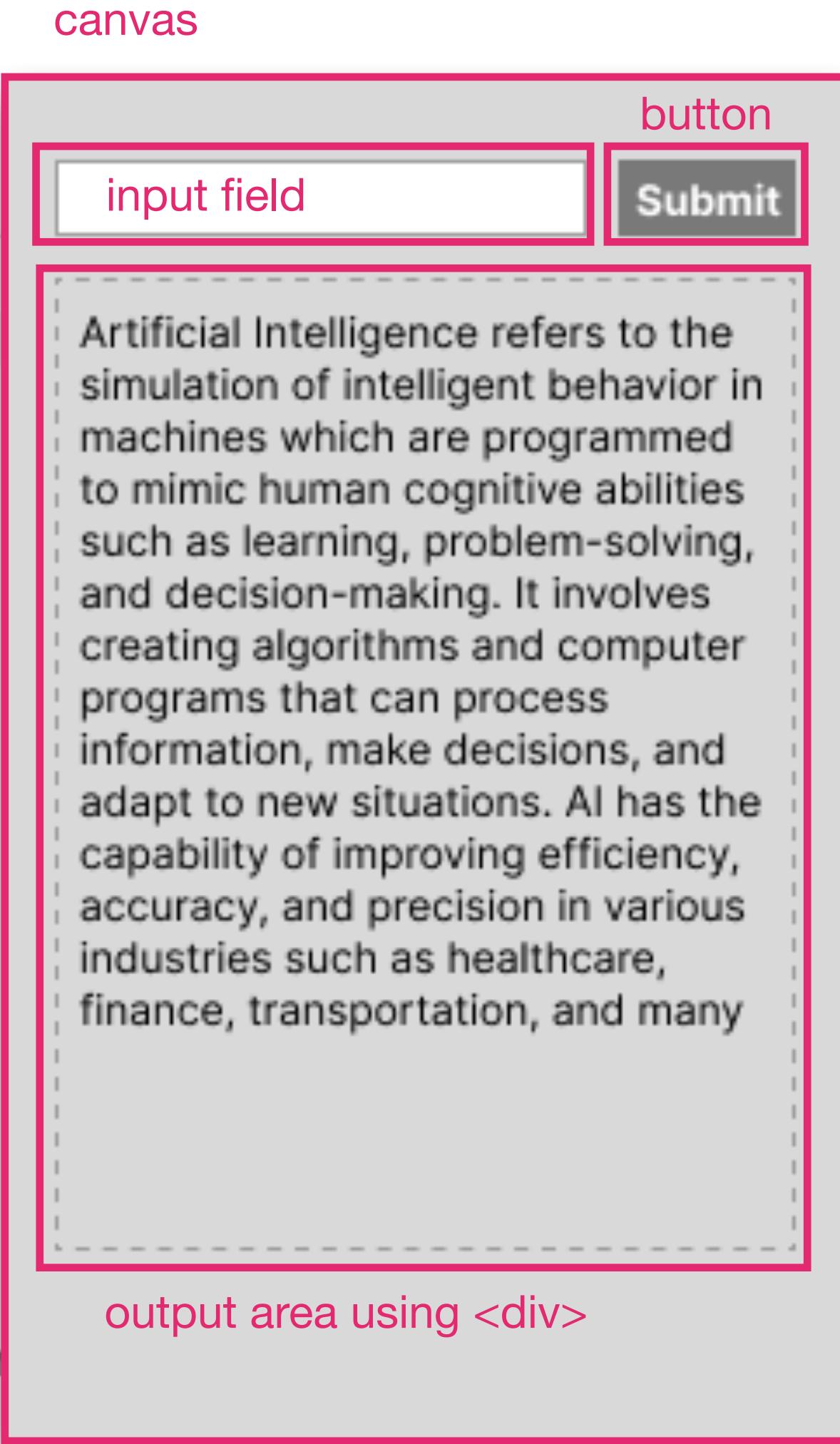
Preview

```
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 OCSI.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 OCSI.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 }
32
33 function draw() {
34   background(220);
35 }
36
37 function buttonEvent() {
38   print("send data to DF");
39   print(input.value());
40   var data = {
41     "input": input.value()
42   };
43
44   OCSI.send("callChatAPI", data);
45 }
```



# Step 1: Create a visual layout of your interface

```
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  OOCXI.connect('wss://oocxi.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.oocxi('returnChatAPI', {text: "gpt gen...  
21  OOCXI.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 }  
32
```



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

## Step 2: add a mouse event

```
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  OOCXI.connect('wss://oocxi.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.oocxi('returnChatAPI', {text: "gpt generated content"})  
21  OOCXI.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}  
32
```

When the button is clicked, then trigger this buttonEvent

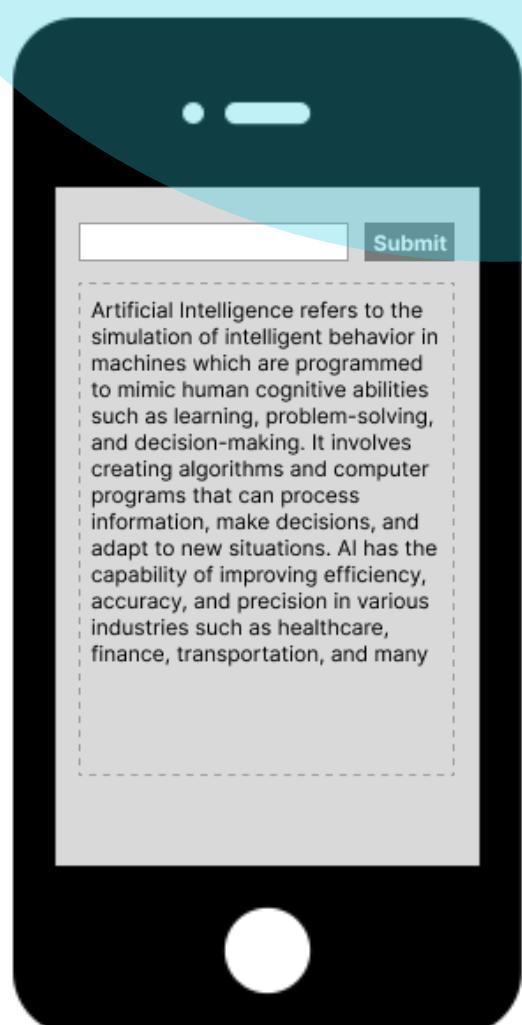
```
36  
37 function buttonEvent() {  
38   print("send data to DF");  
39   print(input.value());  
40   var data = {  
41     "input": input.value()  
42   };  
43  
44   OOCXI.send("callChatAPI", data);  
45  
46 }  
47
```

add a mouseClicked event



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

**OOCSI**



`callChatAI`

**input**

**DF Script**

**script input**

**script output**



**DATA  
FOUNDRY**



**OpenAI**

- Text completion
- Chat completion
- Moderation

**output**

`returnChatAI`

## 4 Models on DF:

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

**p5.js**



**OOCSI**

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

## Step 3: connect to OOCSI on DF

```
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 }
```

# Step 4: send input data to DF via OOCSI channel

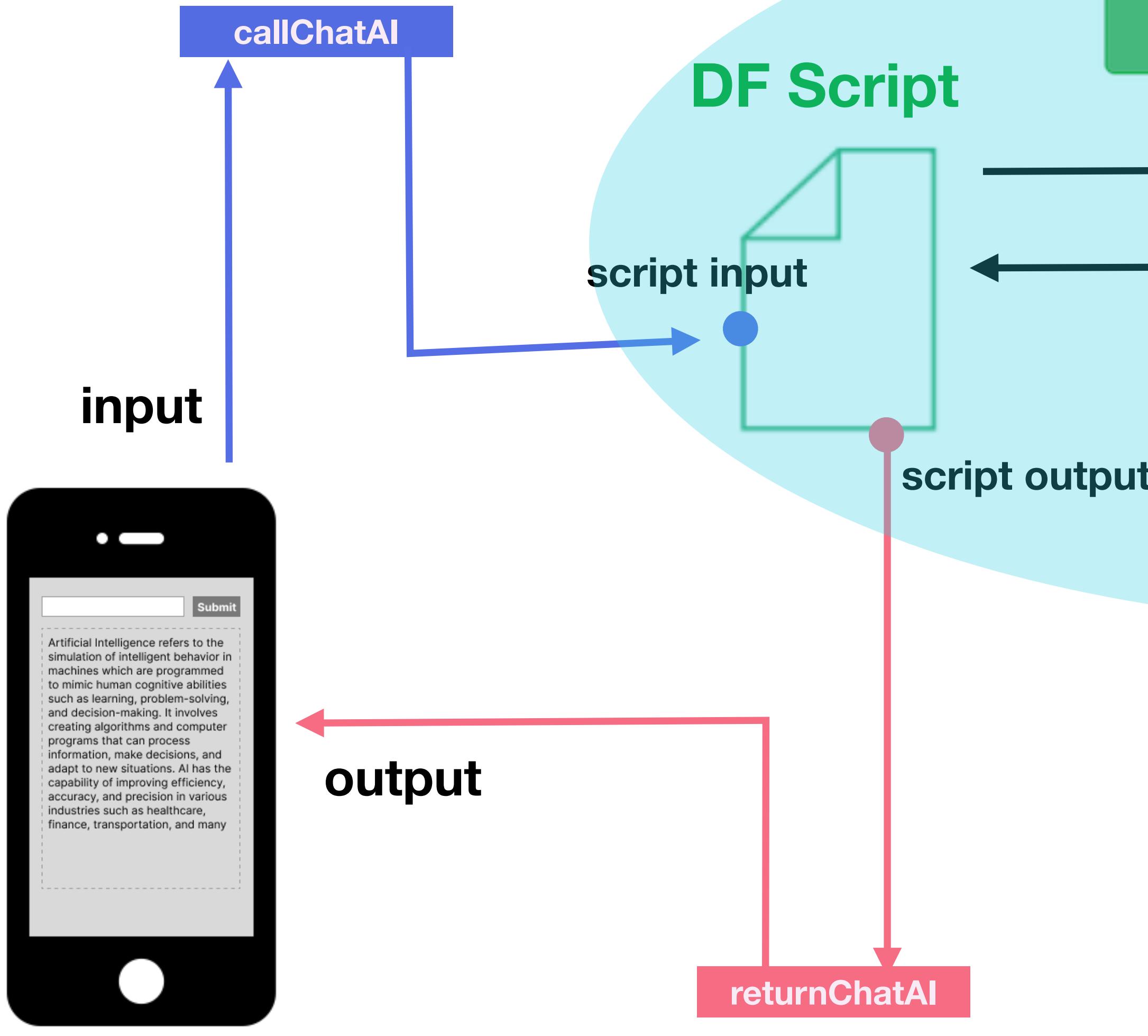
```
1 function setup() {  
2     createCanvas(400, 400);  
3     input = createInput();  
4     input.size(200, 10);  
5     input.position(10, 10);  
6  
7     //create a button  
8     button = createButton("submit");  
9     button.position(220,10);  
10    button.mouseClicked(buttonEvent);  
11  
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}  
32  
33  
34  
35  
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  
48    send data to "callChatAPI" channel
```

A pink arrow points from line 10 to line 44, labeled "buttonClicked, then trigger". A curly brace on the right side groups lines 37 through 47, labeled "send data to "callChatAPI" channel".



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

**OOCSI**



## 4 Models on DF:

- (1) Ada (4 ⚡) —> "ada"
- (2) Babbage (5 ⚡) —> "babbage"
- (3) Curie (20 ⚡) —> "curie"
- (4) Davinci (200 ⚡) —> "davinci"



**OOCSI**

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

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

My projects

Archive

Community

Collaborations

Subscriptions

Explore

Data tools

Guides

Support

## SCRIPT 'TEST-SCRIPT'

2023-03-07 2023-06-07

When you write your script code here, you can make use of the DF object to do stuff on Data Foundry. With this object, you can log data to an IoT or diary dataset, send events to Telegram or OOCSI channels, or retrieve items from IoT and Entity datasets. Find more information on the DF object in the scripts [documentation](#).

Write your code below and test it with input data. When you are happy with the results, install the script on an OOCSI channel or on Telegram to let it automatically react to incoming events.

**Write a script code to use “openai-gpt” api on DF**

**SCRIPT CODE**

```
1 DF.print(data.input)
2 let result = DF.api("openai-gpt", {
3   "api_token": "df-SThkJDBJbGlRdWpvQ0pDT0x6Si9kUGHUMjdGN0x",
4   "task": "chat",
5   "messages": [
6     { "role": "user", "content": data.input }
7   ]
8 })
9 DF.print(result)
10 DF.oocsi('returnChatAPI', {text: result.content})
```

**TESTING** 😊 **LIVE** 🏃

**Install script on OOCSI channel or Telegram**  
Fill in a channel name for the OOCSI system here.  
Events on the channel trigger this script. You can also subscribe to Telegram messages (to build a small chat bot), just enter "telegram" here. Read [more](#).  
Input OOCSI channel or 'telegram' **assign a name for this channel and enable it**

**callChatAPI**

**ENABLE** **DISABLE**

Scripts run at most once per second (3 seconds when accessing/sending data).  
Find the output (of DF.print) below.

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

## 4. Testing Tab: testing your script with input data

### SCRIPT CODE

```
1 DF.print(data.input)
2 let result = DF.api("openai-gpt", {           1. api_token
3   "api_token": "df-SThkJbGlRdWpvQ0pDT0x6Si9kUGHUMjdGN0xDeU0rNnc"
4   "task": "chat", 2. type of task
5   "messages": [
6     { "role": "user", "content": data.input } 3. input data
7   ]
8 }
9 DF.print(result)
10 DF.oocsi('returnChatAPI', {text: result.content})
```

SAVE

OPEN EXAMPLES

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?

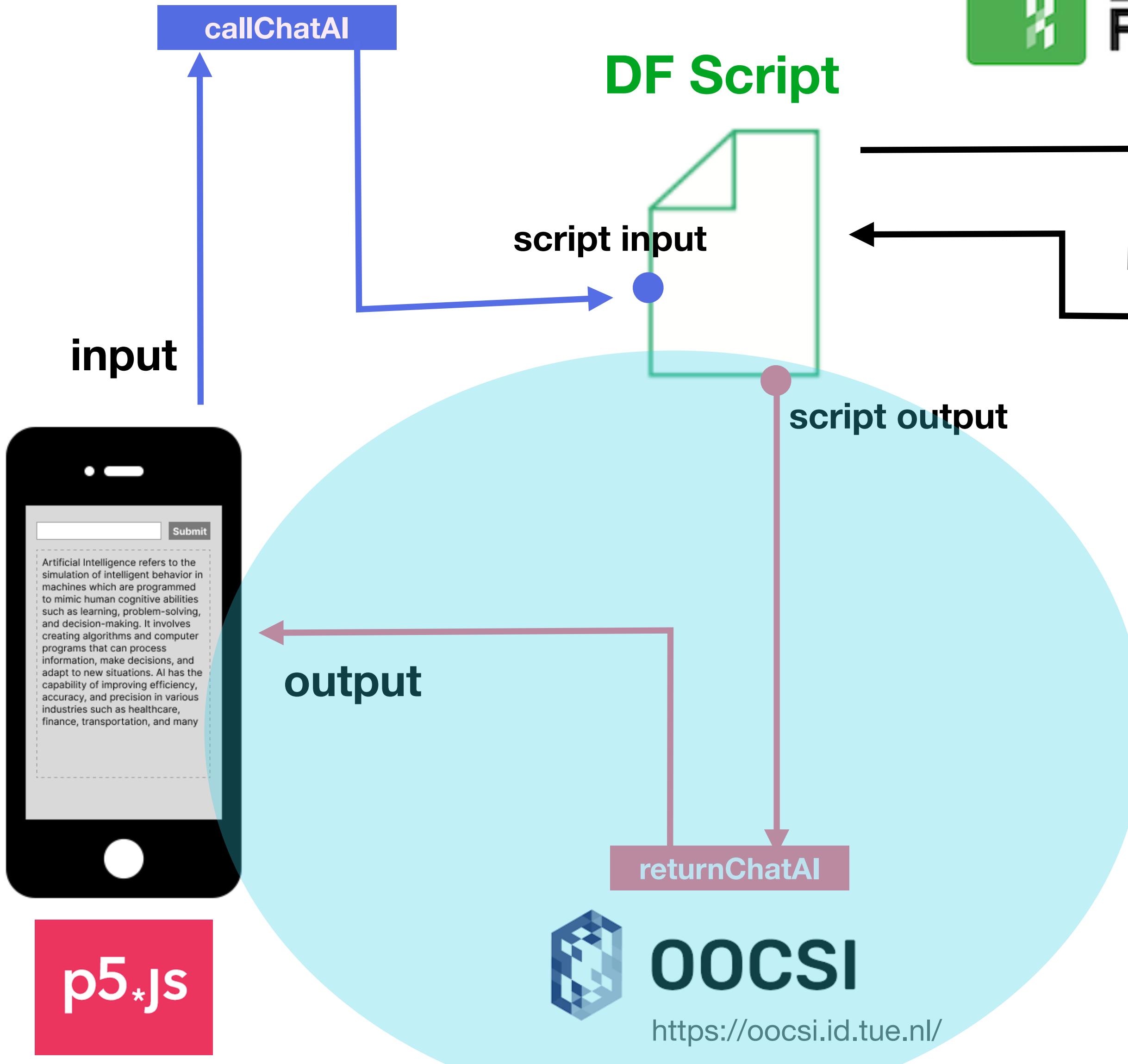
12:18:36 {"role":"assistant","content":"\n\nAI (Artificial Intelligence) is a branch of computer science that focuses on creating machines or software that can perform tasks that typically require human-level intelligence, such as visual perception, speech recognition, decision-making, and language translation.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

## 5. get output using “DF.print”



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

**OOCSI**



## 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

```
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}
```

SCRIPT CODE

DATA FOUNDRY

TESTING 😊

LIVE 🏃

Install script on OOCSI channel or Telegram

Fill in a channel name for the OOCSI system here.

Events on the channel trigger this script. You can subscribe to Telegram messages (to build a small chat bot), just enter "telegram" here. Read [more](#).

Input OOCSI channel or 'telegram'

callChatAPI

ENABLE

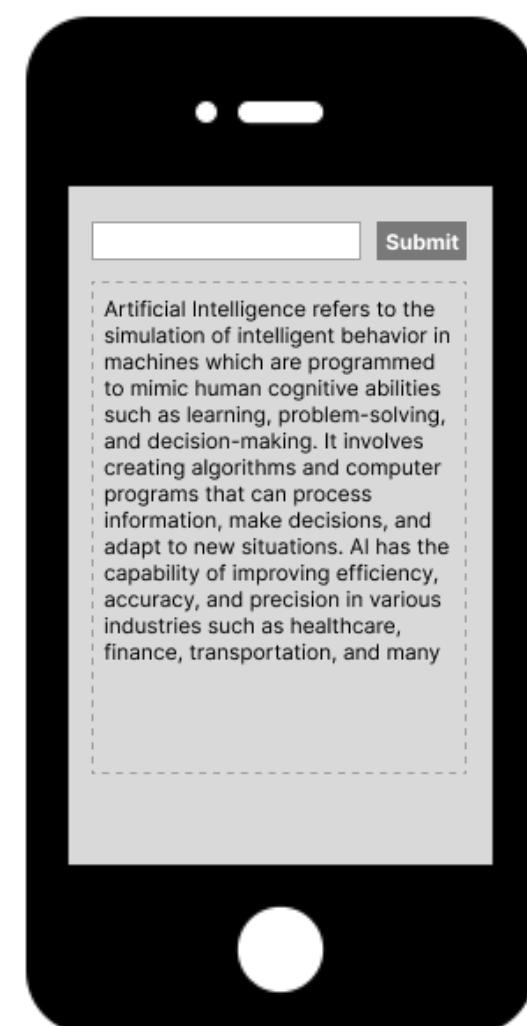
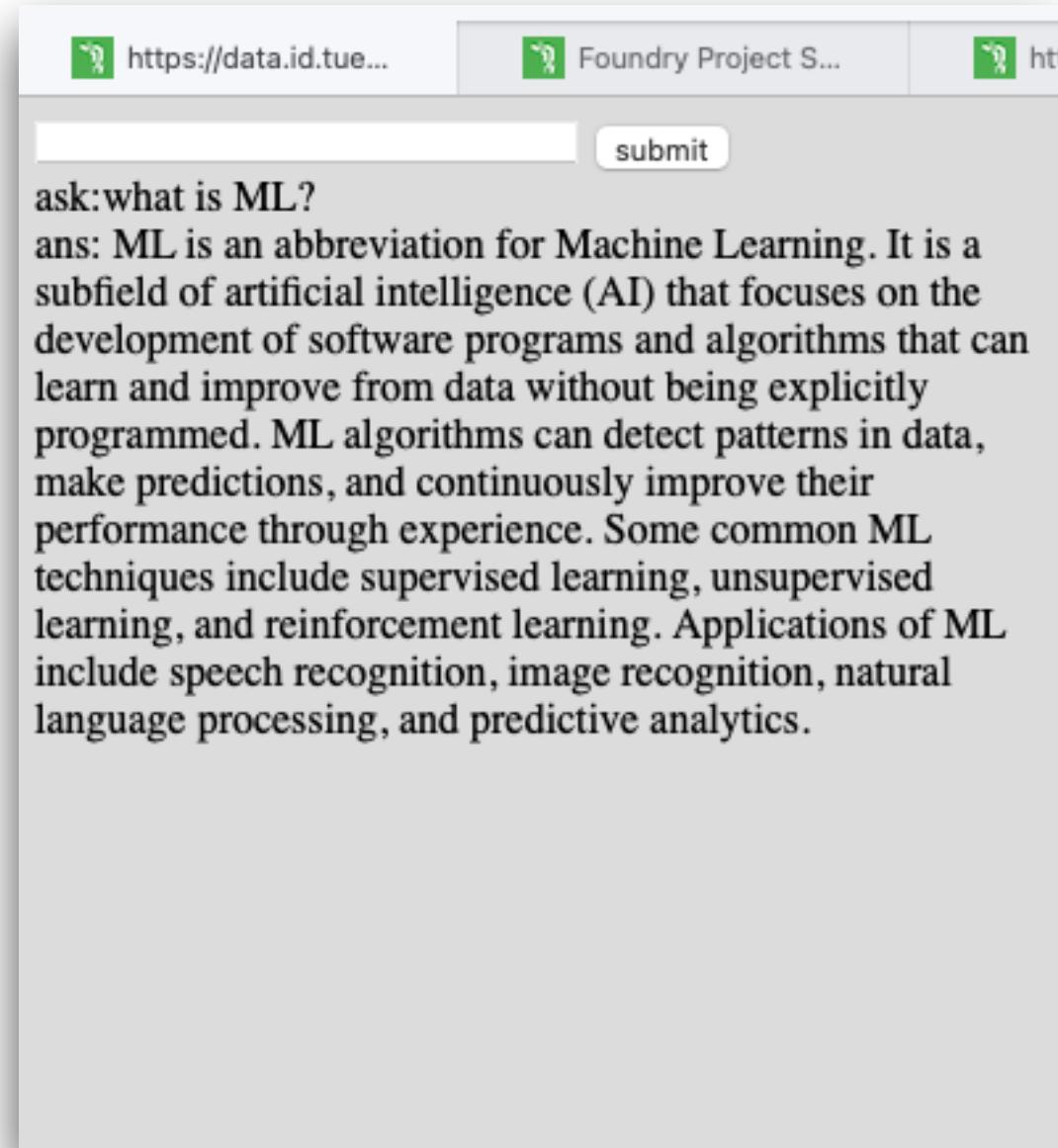
DISABLE

send data to "returnChatAPI" channel



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

**OOCSI**



**DATA  
FOUNDRY**



**OOCSI**

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

**callChatAI**

**input**

**DF Script**

**script input**

**script output**

**output**

**returnChatAI**

**OpenAI**

- Text completion
- Chat completion
- Moderation

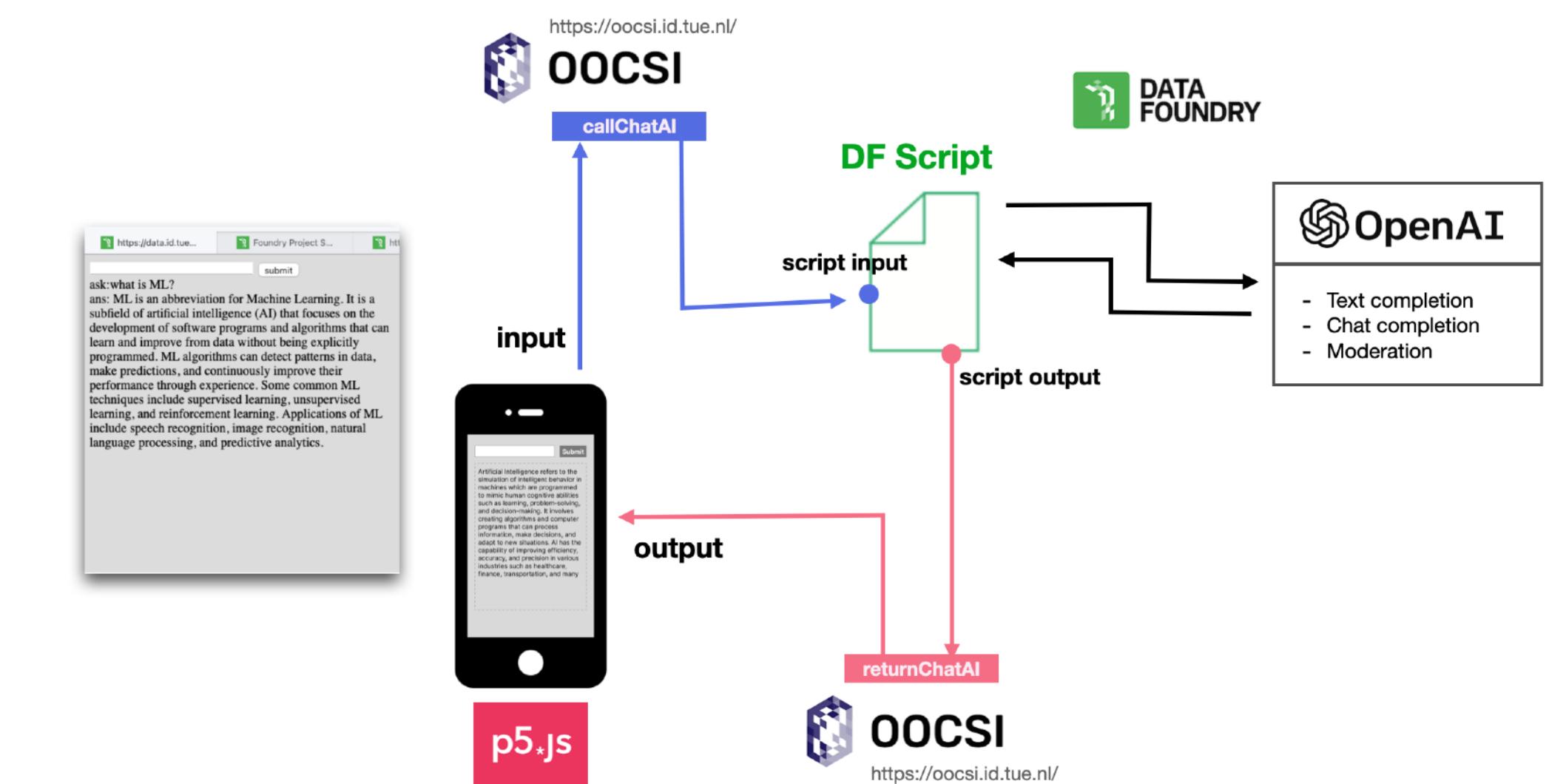
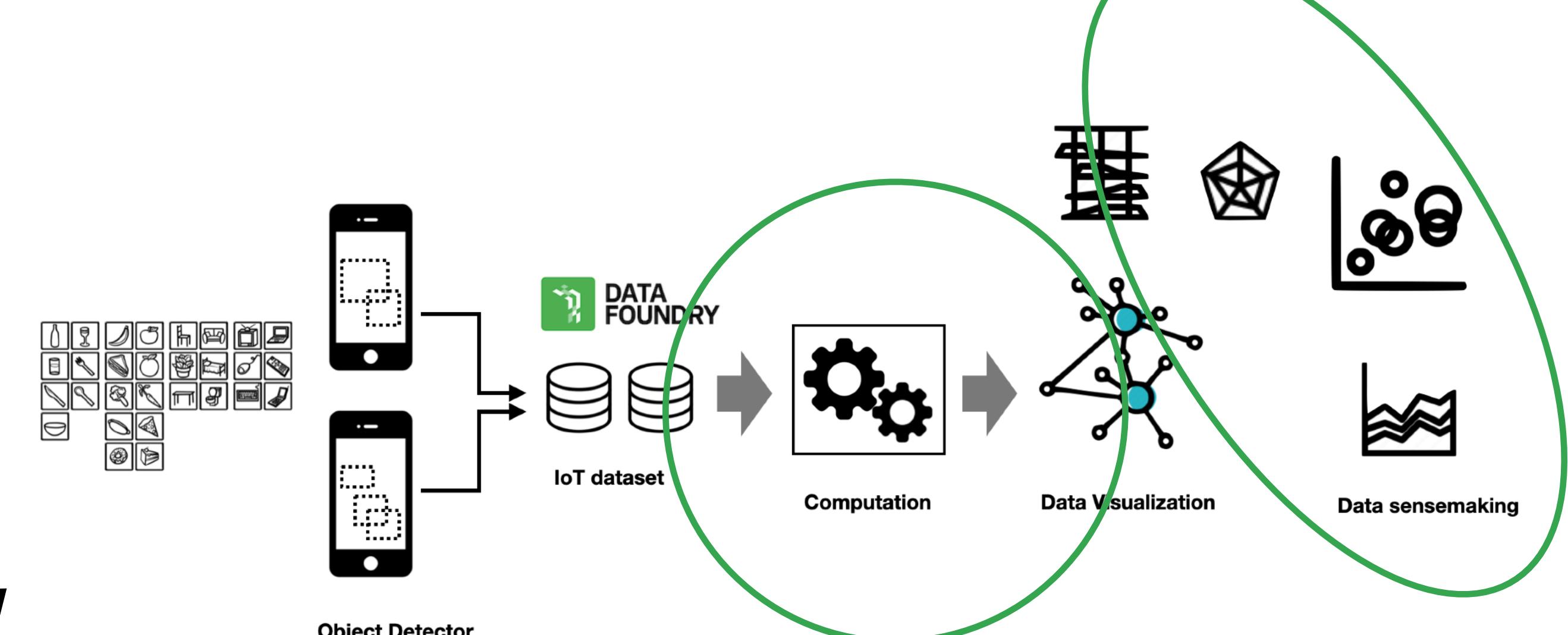
## MDC Workshop (I): Advanced (2 hrs)

### 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**