

# Using the BEAM Notation

## Setup

The boxology modeler is based on draw.io, a popular diagramming tool. Here are the necessary steps to prepare for modeling your solution:

### 1. Install or access draw.io

To use draw.io, you have two options:

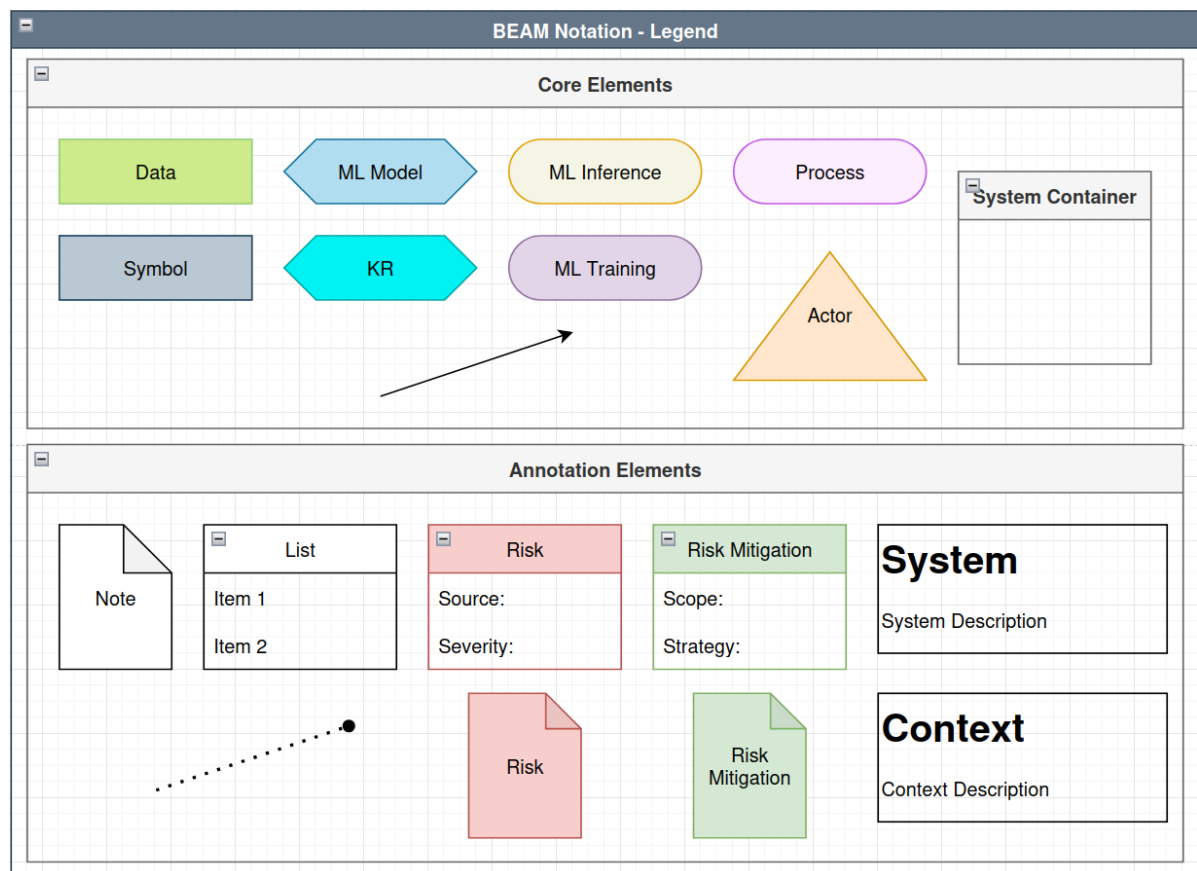
- a) Use the browser version at <https://draw.io>
- b) Download and install draw.io Desktop from <https://get.draw.io>

### 2. Import the BEAM notation library [1] that contains all the graphical elements

- a) Download the BEAM notation library in version 3 (beam\_lib\_v3.xml)
- b) Import the library (both desktop and online version): File > Open Library from > Device > [location of your downloaded file beam\_lib\_v3.xml]

### 3. Start modeling

- a) Access the library beam\_lib\_v3 on the top-left of your draw.io application in the shapes sidebar
- b) Start by adding the “Legend” component to your drawing by clicking it



- c) You can now draw from scratch by copy-pasting elements from the legend or by clicking the needed components in the shapes sidebar (top-left in the draw.io application)

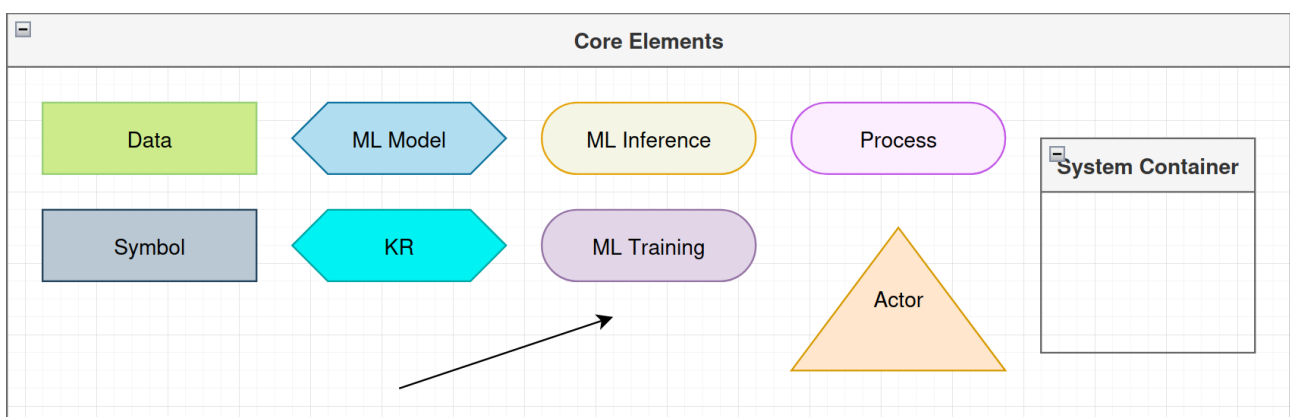
#### 4. Save and submit

- a) Save your file (\*.drawio or \*.drawio.xml)
- b) Submit your file [via Canvas](#)

## Modeling

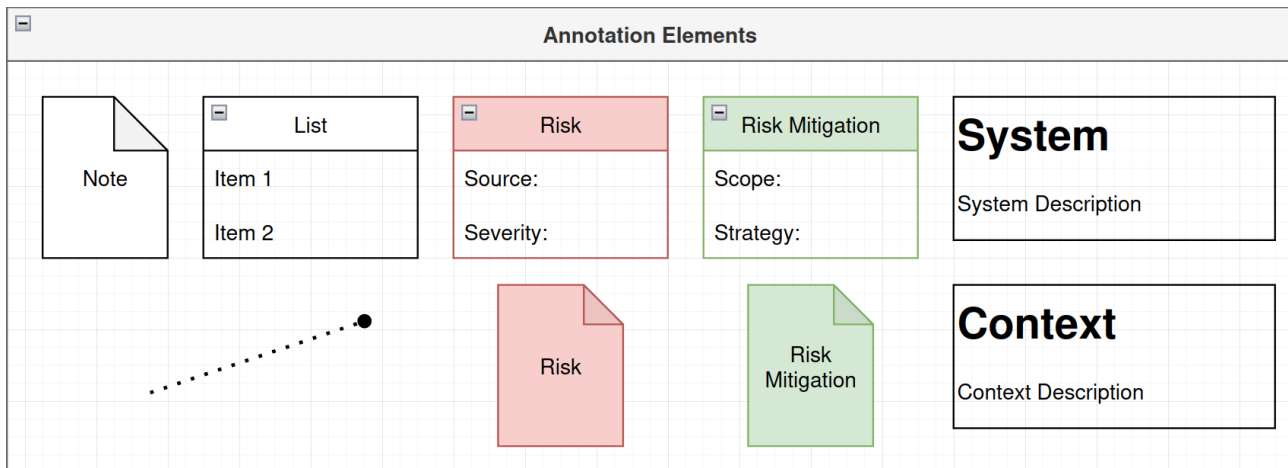
In the following, we will (briefly) explain the elements provided in the library (the grouping of the components is shown in the “Legend” box). It consists of

1. **Core drawing elements** used to represent the building blocks of your Data Science pipeline, including
  - a) Input/Output **Data** (green box), e.g. for text, tabular data, images, ...
  - b) **Symbolic** Input/Output Resources (grey box), e.g. for knowledge graphs, ...
  - c) **ML Model** (blue hexagon)
  - d) **Knowledge Representation and Reasoning (KR)**, turquoise hexagon), e.g. for an ontology reasoner
  - e) **ML Inference** (light orange rounded rectangle)
  - f) **ML Training** (purple rounded rectangle)
  - g) General **Process** Elements (pink rounded rectangle), e.g. for data preprocessing
  - h) **Actor** (orange triangle), e.g. an engineer or user
  - i) **System Container**, for scoping different parts of the system
  - j) **Solid black arrow** to draw the connection/workflow between core components

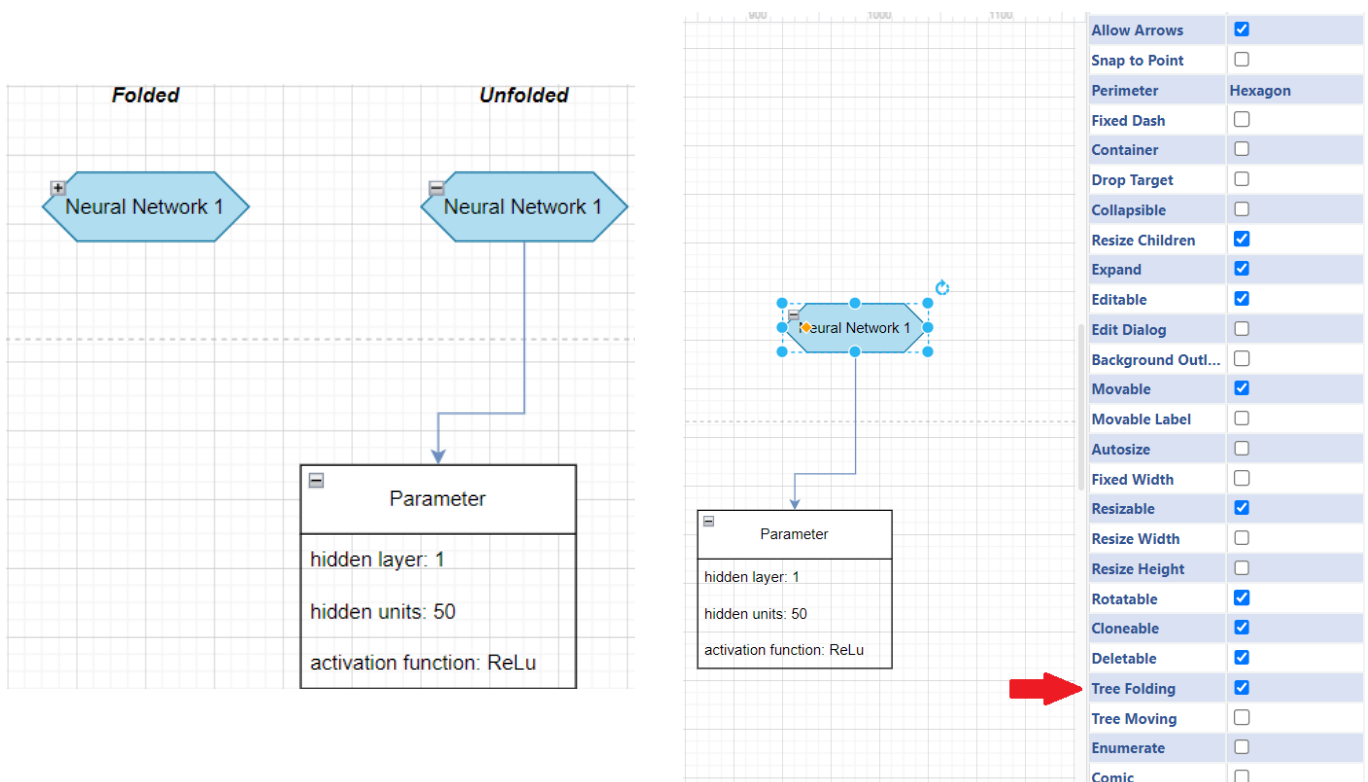


2. **Annotation drawing elements** used to annotating your Data Science pipeline building block, including
  - a) **Note** to represent any free-text annotation
  - b) **List** to represent list or key-value pair annotations

- c) **Risk** (as free-text or list) to represent risks associated with components or the system
- d) **Risk Mitigation** (as free-text or list) to represent risk mitigation for risks
- e) **System** and **Context** description to provide textual information of the system as a whole and the context it is operated in
- f) **Dotted arrow** to associate any of the above annotation elements to the relevant component



3. **Show/hide details:** In case the model contains details that should not always be visible, e.g. to hide them for an overview, this can be accomplished using the **Tree Folding** property. The attached annotation details of elements with this property can be collapsed/expanded using the small minus/plus-icon on the element. Note: Unfortunately, draw.io does not support this for annotation details directly attached to container elements.



## Example

We provided an example of a system in the BEAM notation, which is described below and can be downloaded at [2].

The depicted system is a classifier of fashion attire images. It is used in a recommender system of an online shop that sells fashion items. Users can upload images of fashion attire they like in order to receive similar recommendations.

At the heart of the system is a machine learning model, which is trained on a dataset of labeled fashion attire images. The model is trained to classify which type of fashion attire an image shows, e.g. T-shirt, pullover, ... The BEAM notation depicts the training and inference processes separately.

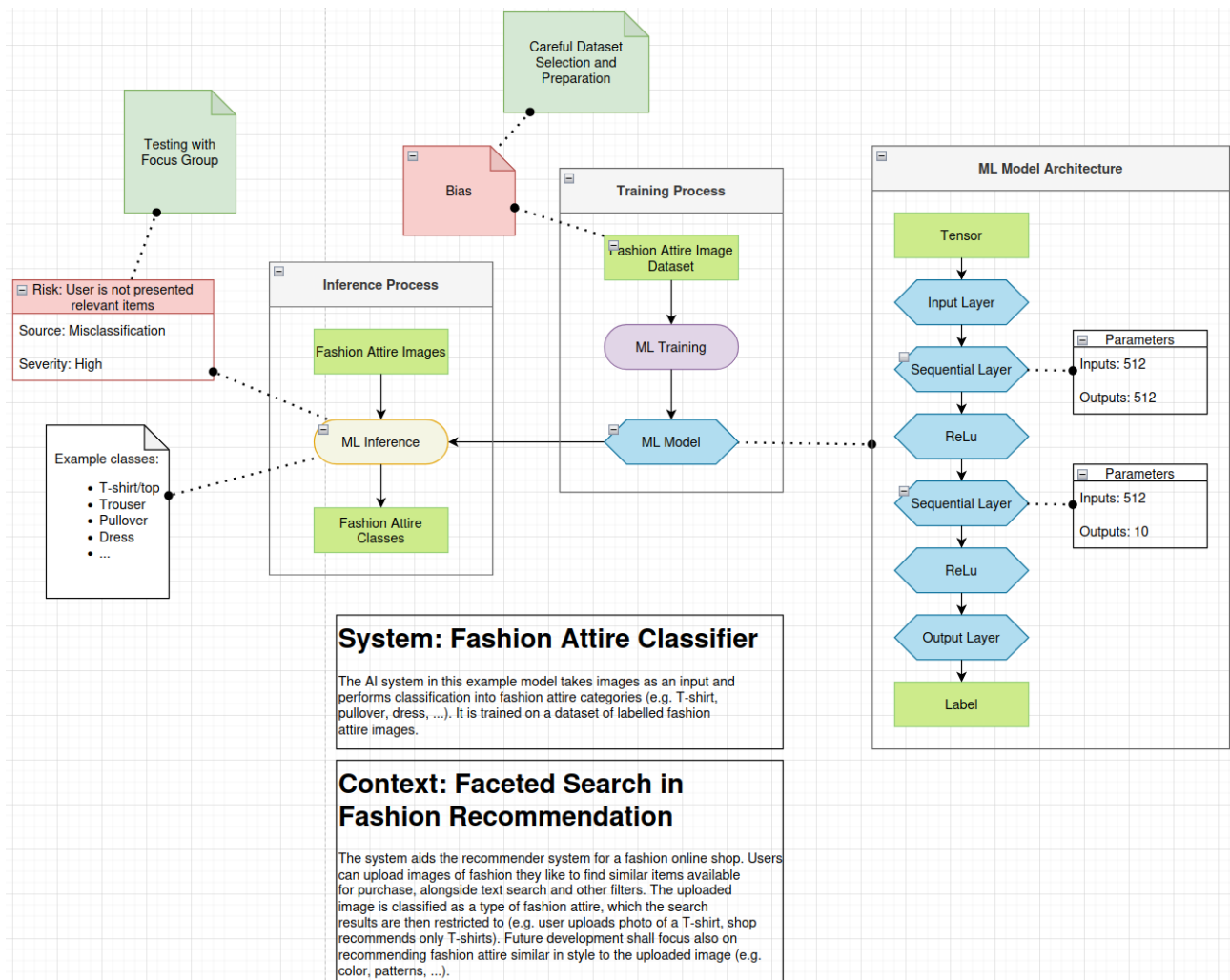
In the training process the aforementioned dataset is used to train the model. The internal model architecture is also depicted (an image is fed into the input layer of the ML model, with data flowing through the various layers of the model until the final layer produces a label as its output).

In the inference process, the ML model is used to classify the input images.

Two risks associated with this system are also modeled in the example. First, the dataset might contain some bias (for example there might be a disproportionate amount of pink dresses, which might cause the model to be biased towards classifying images of pink objects as dresses). This risk can be mitigated by selecting a high-quality dataset and/or additional preparation of the dataset.

The other depicted risk is that customers of the online shop might not be shown relevant items if the uploaded images are classified incorrectly. The mitigation strategy for this risk is to test the system with the trained model with a focus group of potential customers and evaluate the results.

The example furthermore includes a system and context description to provide a quick overview.



## References

[\*] A more comprehensive documentation is available in the following link:

[https://github.com/wu-semsys/beam\\_tutorial/blob/main/beam\\_technical\\_documentation\\_v3.md](https://github.com/wu-semsys/beam_tutorial/blob/main/beam_technical_documentation_v3.md)

[1] [https://github.com/wu-semsys/beam\\_tutorial/blob/main/beam\\_lib\\_v3.xml](https://github.com/wu-semsys/beam_tutorial/blob/main/beam_lib_v3.xml)

[2]

[https://github.com/wu-semsys/beam\\_tutorial/blob/main/beam\\_v3\\_example\\_fashion\\_attire\\_classification.xml](https://github.com/wu-semsys/beam_tutorial/blob/main/beam_v3_example_fashion_attire_classification.xml)