Using the BEAM Notation

This document shall provide quick-start information for using the BEAM notation. This document is structured as follows:

1. Setup
2. Modeling
3. Modeling Risks and Mitigation
4. Example
5. Dos and Don’ts – Best Practices
6. References

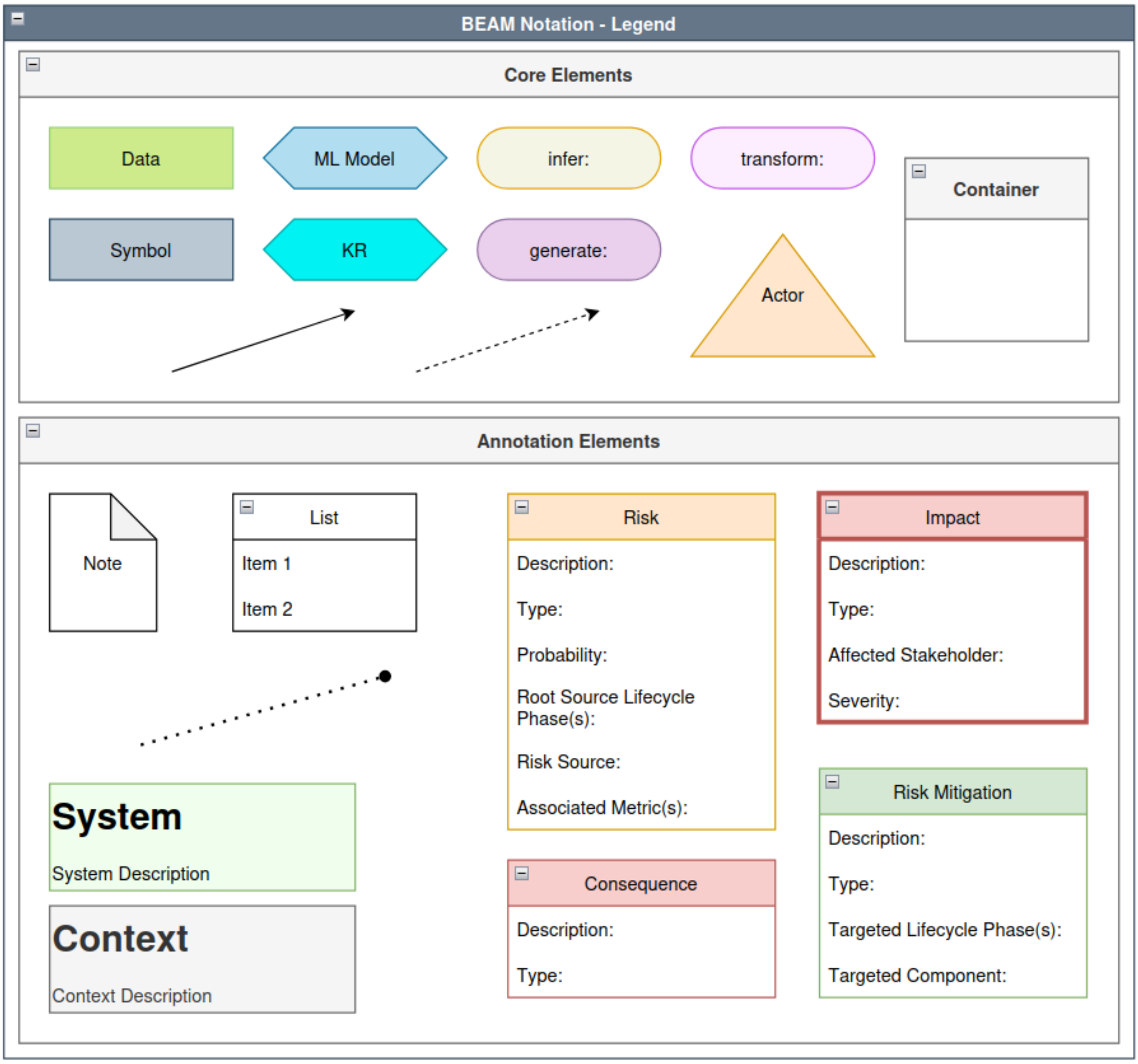
1. Setup

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

2. 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) **Symbol**ic 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) **Inference Process** (light orange rounded rectangle, prefix “infer:”), e.g. “infer: deduce” for deductive inference processes in case of using an ML model for classification or running an ontology reasoner

f) **Generation Process** (purple rounded rectangle, prefix “generate:”), e.g. “generate: train” for ML training processes or “generate: engineer” for knowledge engineering

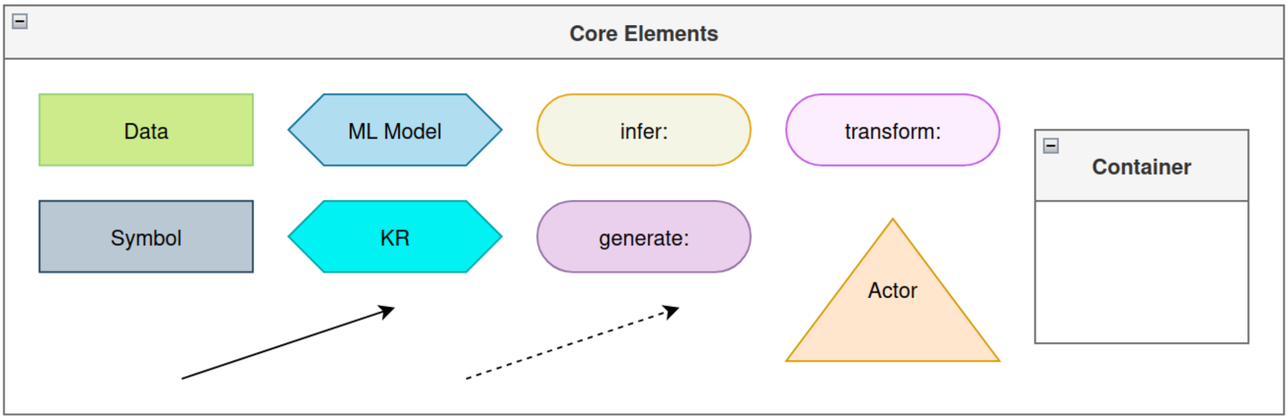
g) **Transformation Process** (pink rounded rectangle), e.g. for data pre-processing or mapping, no new knowledge is generated, these processes are strictly transformational

h) **Actor** (orange triangle), e.g. an engineer or user

i) **Container**, for scoping different parts of the system

j) **Solid black arrow** to draw the connection/workflow between core components

k) **Striped black arrow** to represent auxiliary inputs, i.e. data/symbolic input that is not necessary for the data flow, but will be used as additional input if available



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** to represent risks associated with components or the system

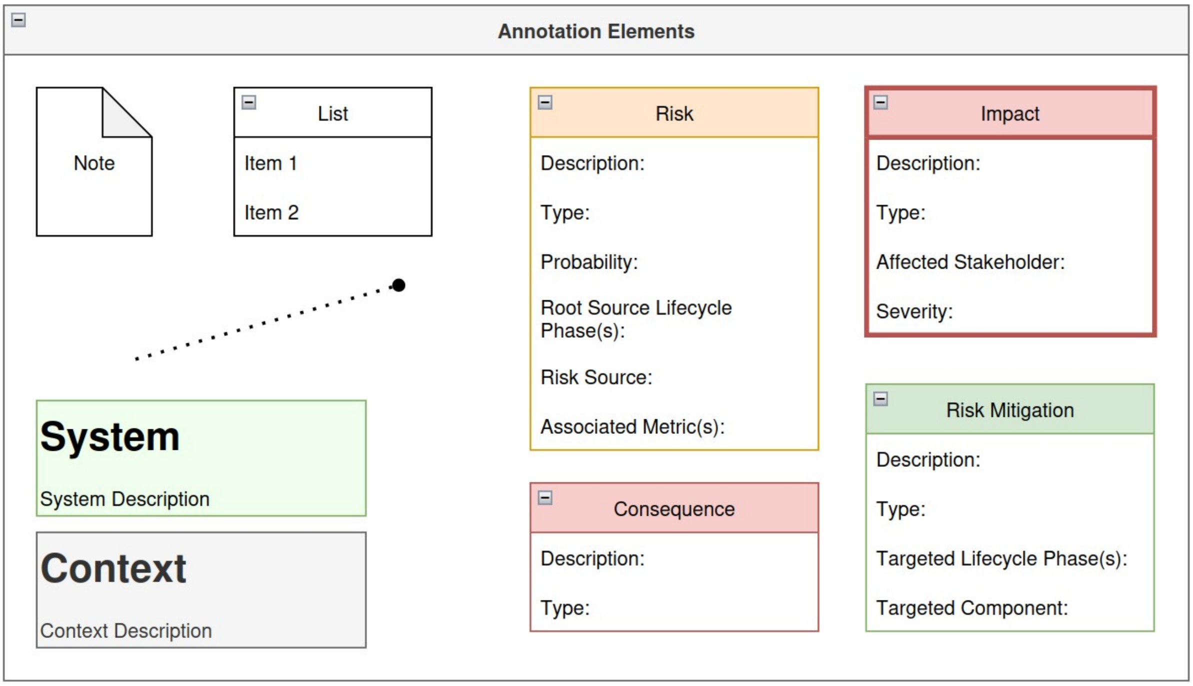
   d) **Consequence** to represent the consequence of a risk if it materializes

   e) **Impact** to represent the impact of risks/consequences on specific stakeholders/groups

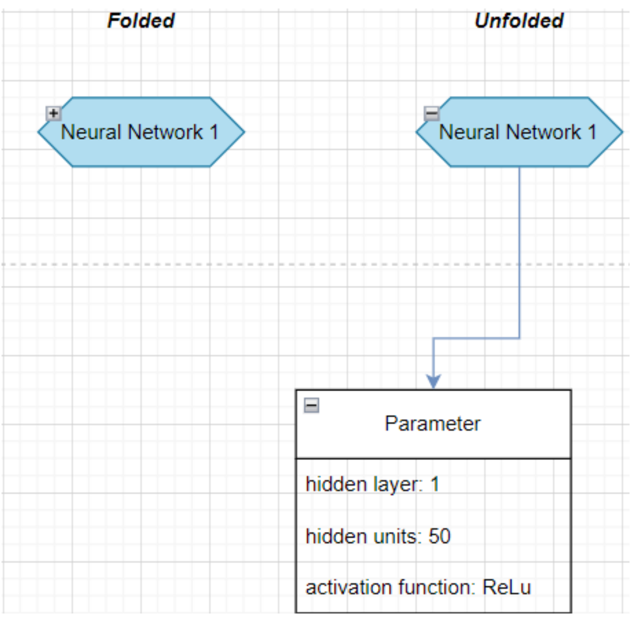
   f) **Risk Mitigation** to represent risk mitigation for risks

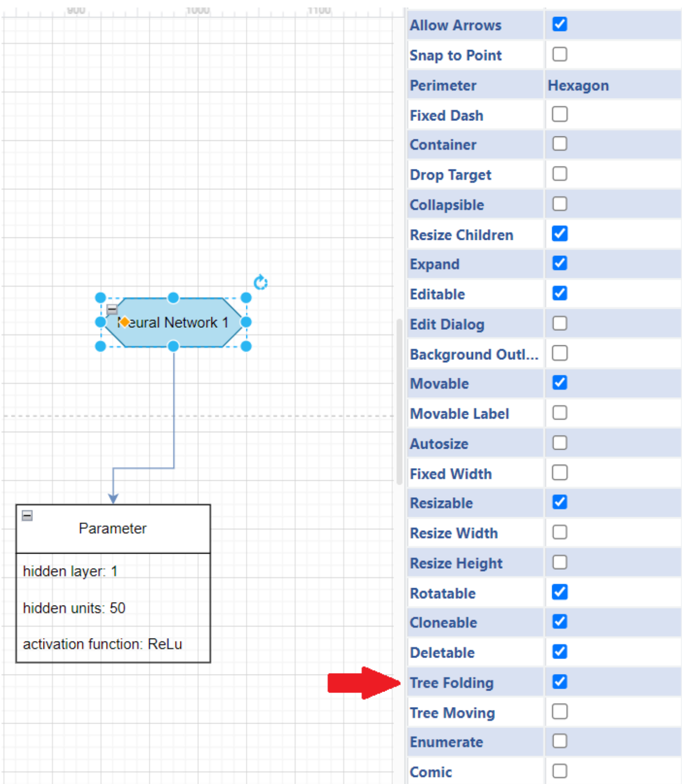
   g) **System** and **Context** description to provide textual information of the system as a whole and the context it is operated in

   h) **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.



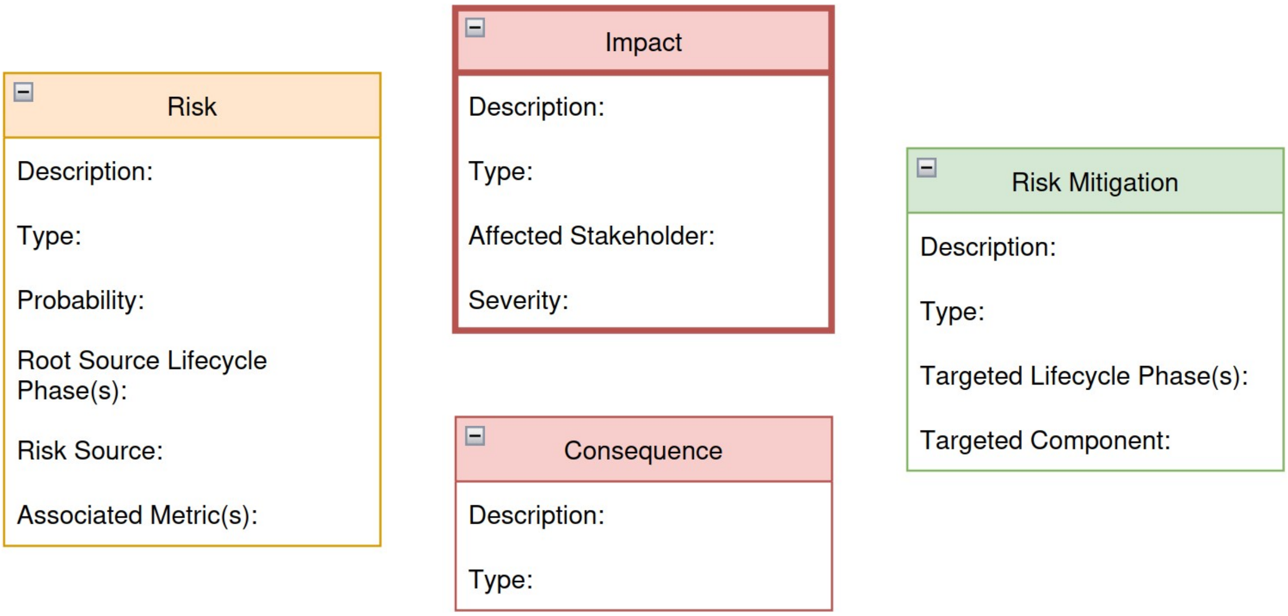


3. Modeling Risks and Mitigation

As described in the previous section, BEAM includes special annotation elements for representing risks and their consequences, impacts and mitigation strategies.

Risk elements are orange, consequence and impact elements are red (with impact elements having a thicker border) and risk mitigation elements are green. They all are list-style elements with attributes in the form of key-value pairs.

Guidelines and examples for the attributes of the risk, consequence, impact and mitigation elements will be provided.



Risks can be associated to one or more elements using annotation connectors, this includes container elements. System level risks can represented by attaching risk elements to an overarching container that includes the whole system.

Consequence elements are used to model consequences in case a risk materializes. Impact elements are similar to consequence elements, but they always have an affected stakeholder. Risks, consequences and impacts are to be connected with regular workflow connectors (solid black arrow) to represent their causal relationship.

Example: In a system that relies on an LLM, there is the risk of a malicious prompt (Risk), which can lead to offensive behavior of the model (Consequence), which then affects different stakeholders (Impacts), e.g. certain user groups are discriminated and the system provider suffers loss of reputation.

Risk mitigation elements can also be associated to one or more elements using annotation connectors in the same way as risks (or other annotations). Often a risk mitigation measure will be associated to a specific risk, consequence or impact, in which case this connection should also be modeled using an annotation connector. Risk mitigation elements, however, do not have to be connected to risk, consequence and impact elements, and they can also be connected to both risk/consequence/impact elements and other elements of the system at the same time.

4. 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.

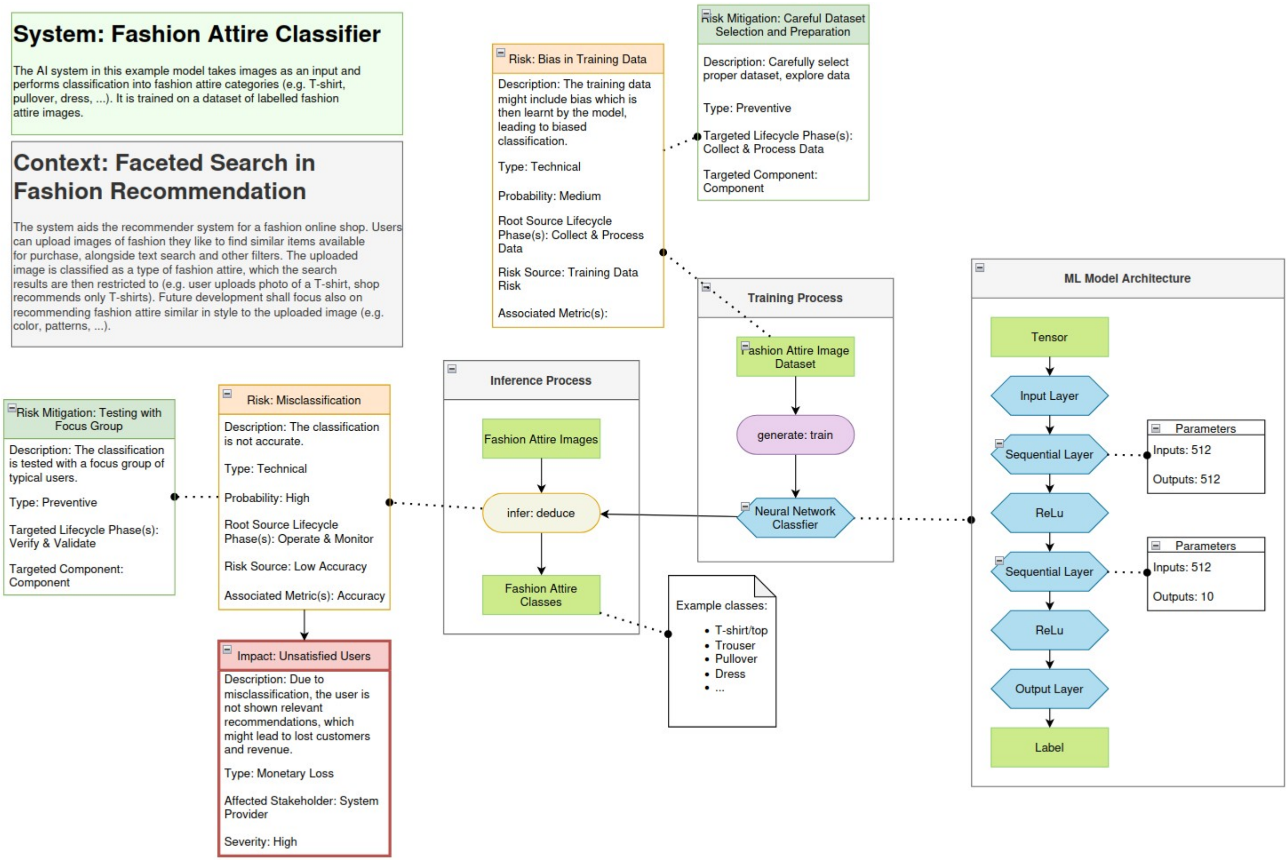
At the heart of the system is a machine learning model, which is trained on a dataset of labeled fashion attire images. The model is trained to classify which type of fashion attire an image shows, e.g. T-shirt, pullover, … The BEAM notation depicts the training and inference processes separately.

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. As this might turn away customers, monetary loss for the owner of the online shop (system provider) is modeled as an impact of this risk. 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.

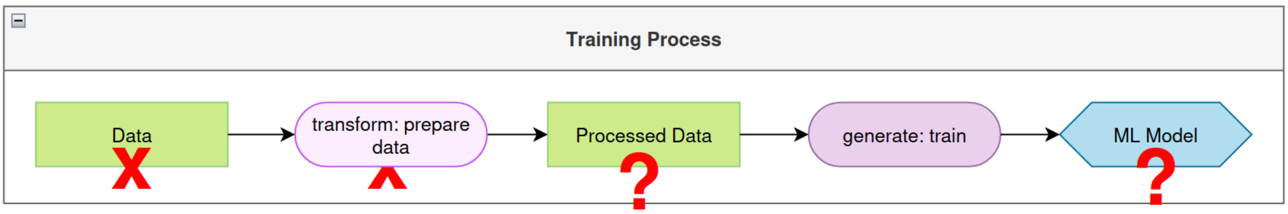
The example furthermore includes a system and context description to provide a quick overview.



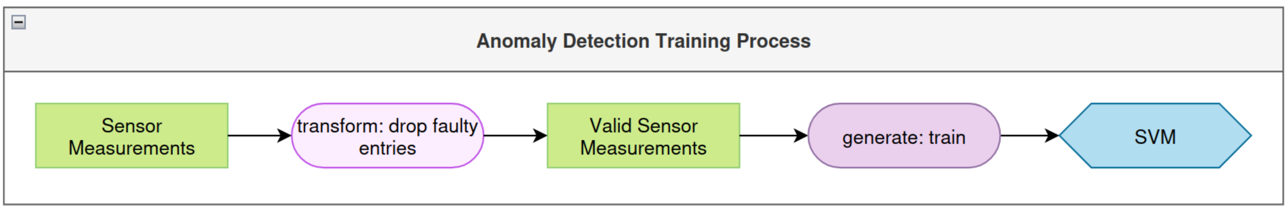
5. Dos and Don’ts – Best Practices

This section gives a few small examples with mistakes and how to avoid them.

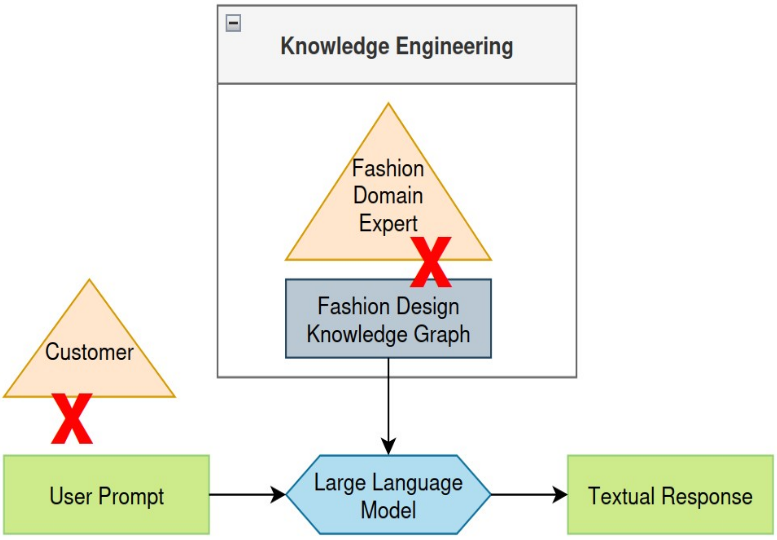
5.1 Example 1 – Unspecific Labels



In this example, very generic labels are used, s.t. the details of the modeled training process remain unclear. What type of data is used? Where does it come from? Data Preparation is too broad of a label, which concrete steps does the data preparation process consist of? Could the processed data be described more specifically? What kind of model is trained here? Below is the same example with more descriptive labels:

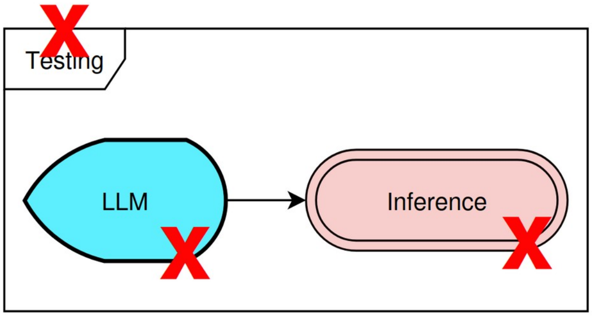


5.2 Example 2 – Missing Connectors



In this example, the actors are not connected to the rest of the model. Since BEAM models shall be automatically processable, the actors should be connected to the relevant elements using workflow connectors. Co-occurrence in the same container or proximity to the relevant elements are not enough – the workflow must be explicitly modeled.

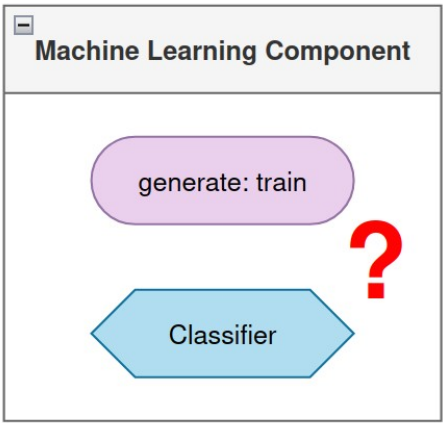
5.3 Example 3 – Unknown Shapes



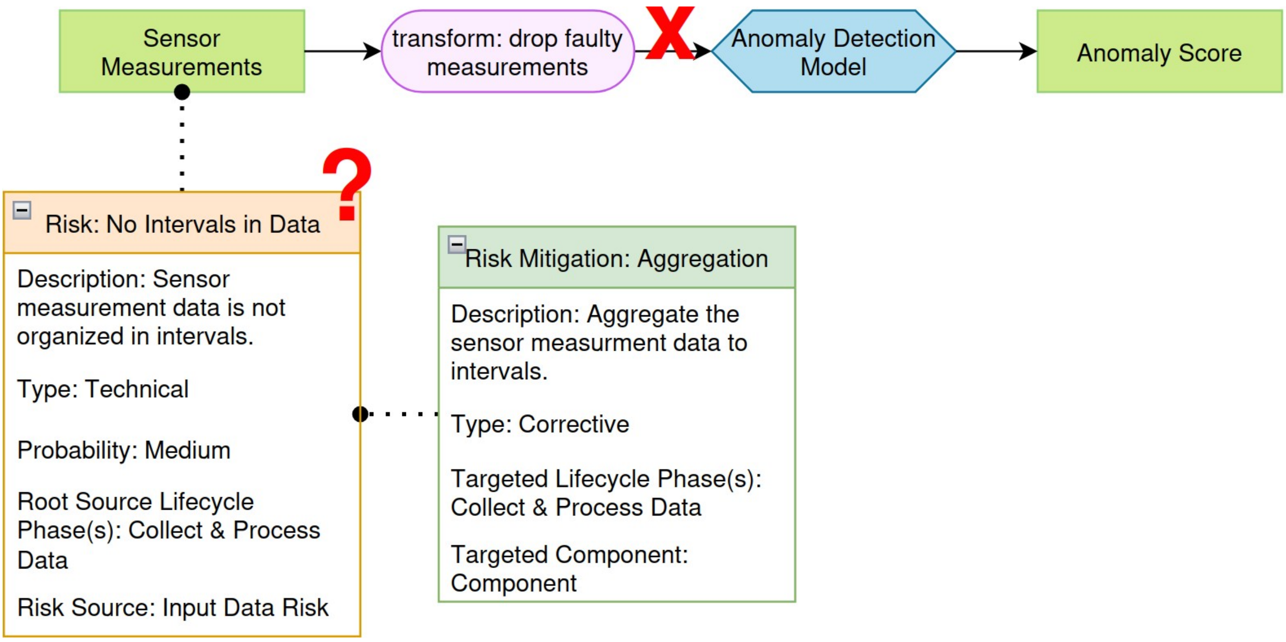
In order for BEAM models to be understandable for humans and automatically processable, elements that are not from the library shall not be used. This includes elements from other well-known modeling languages, such as UML.

5.4 Example 4 – Missing Elements and Connectors

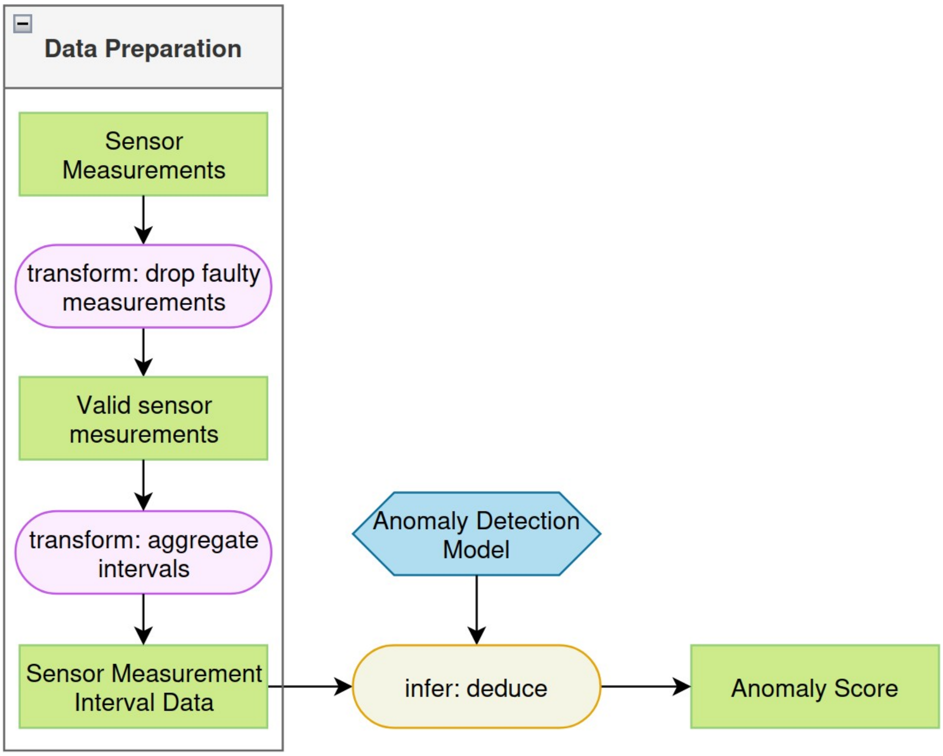
As mentioned in a previous example, an explicit connection between the training process and the classification model is missing. Furthermore, this example is missing input to the training process – which data is the classifier trained with?



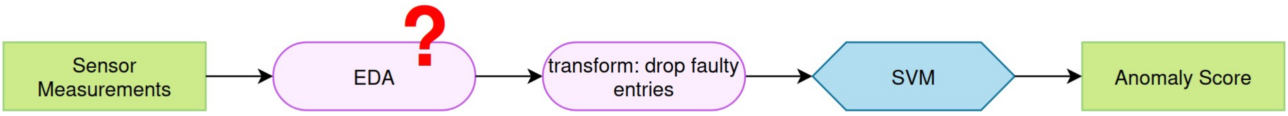
5.5 Example 5 – Missing Elements & Risk vs. Process



There are multiple issues in this example. Firstly, in case the input data (Sensor Measurements) is not always aggregated in a useful way, but there is an aggregation step, this should be modeled as a process and not as a risk with a mitigation strategy. Secondly, a process for data preparation (Drop faulty measurements) connects directly to the model and it is only implied that there is an inference process. The example below correctly represents the inference process and intermediary data output.



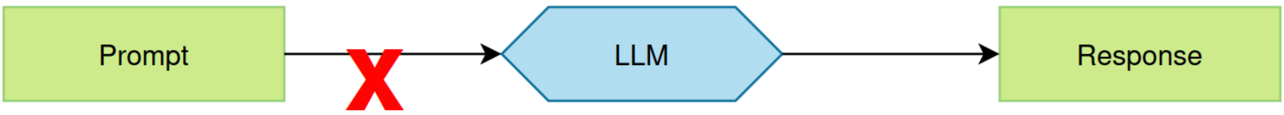
5.6 Example 6 – Represent AI system, not Engineering Process



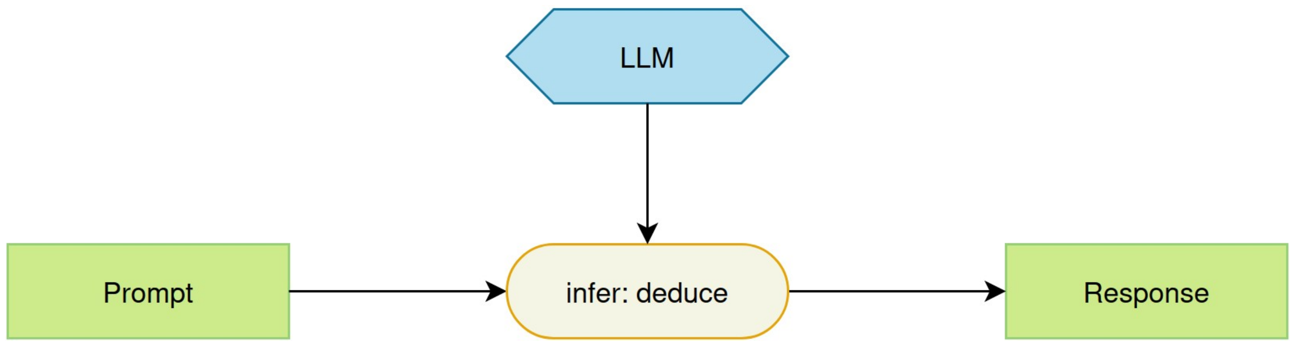
In this example, exploratory data analysis (EDA) is included as a process in the workflow. BEAM is intended to model AI systems, not the engineering process behind it. It might also be helpful to ask what the output of a process is and to model it explicitly. In the case of EDA, the results might be descriptive statistics, data visualization and insights gained about the data – is this used in the AI system or in the engineering process?

Furthermore, the mandatory process type/prefix is omitted for the EDA process – perhaps because neither transform nor infer or generate were a good fit – a possible hint that it might not belong in the BEAM model.

5.7 Example 7 – Data Flow through Model



In this example, the data flow is modeled to go through a model. This is not the intended use of the BEAM notation – rather, the dat flow should go through a process, in which a model is used, see below.



6. 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

[1] 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_classfication.xml