Introduction And Short Overview

Originally, the prediction model was only able to recommend the next component that should be added to **strong** the *italic* construction, *but* `adskfaks #footnote[a] ldjf` *not* where to add it. This project explores a possible modification of the preexisting architecture in order to change that.[1]

ayo

Hello *there*

On a high level, there are two major changes. First, the new model now has another output that is tasked with predicting the node onto which the next component should be added, making it a multi-task model. Second, the component classification now happens per node, as opposed to the per graph classification that was previously done, meaning the model predicts for each node what the next component could be. The final model prediction is then determined by combining both outputs.

The following sections will detail how the dataset is created, have specifics about the architecture and briefly show the findings.

System

The most important files are located in `enco_prediction_scripts/` and `models/enco_prediction/`, in the branch `feature/add-ext-node-pred`.[2].

Dataset

This project was only tested on the `festo_fdt` dataset. There are two steps in creating it:

- `create_enco_partial_graphs.py`: The actual creation of the partial graphs (objects of graph class specified in `preprocessing/graph.py`) with their labels (dictionary containing target node and component, objects of classes in `preprocessing/{node,component}.py`). The resulting (graph, label) tuple list is pickled and stored in `enco_festo_fdt_partial_graphs_{train,val,test}.dat`. A single (graph, label) instance looks like this:

  ```
  (graph, {
    "target_node": <the target node object of the graph>,
    "target_component": <the target component object of the graph>
  })
  ```

  This format was chosen so that the data is easily interpretable, even only on its own.

[1]ayo

[2]enco is an acronym for **E**xtension **N**ode**Co**mponent