# Per Entity Training Pipelines in Apache Beam

Jasper Van den Bossche
ML6

BEAM SUMMIT
NYC 2023

# About ML6

We are a group of AI and machine learning experts building custom AI solutions.

Amongst our engineers we have several Apache Beam contributors.

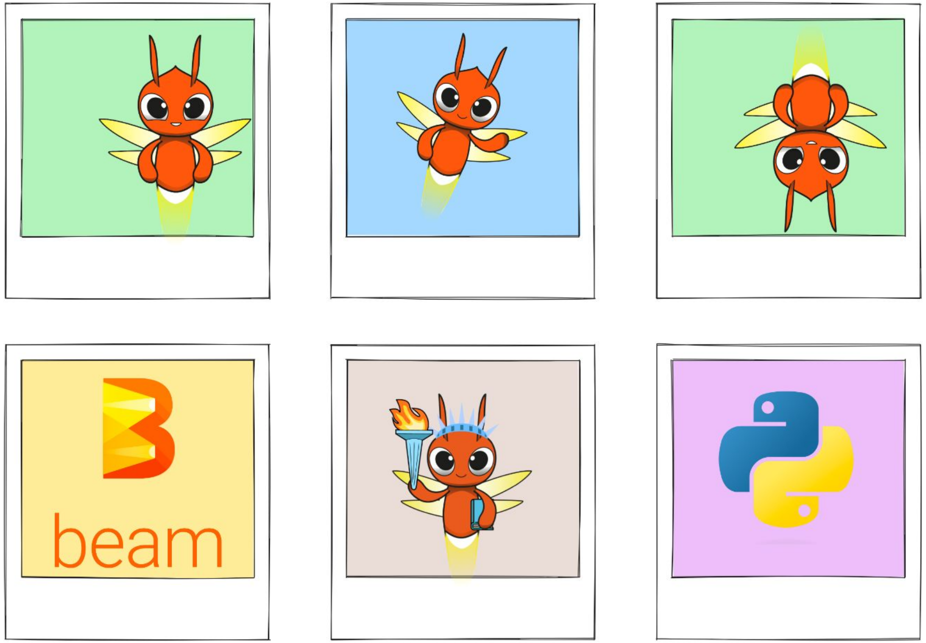
# Agenda

- Development of ML applications
  - What is training?
  - What is MLOps?
- What does per entity training mean?
  - Training multiple models rather than a single model?
  - Why use a per entity strategy
- Example per entity training pipeline
- Bonus: Using trained models in a RunInference pipeline

# What is machine learning model training?

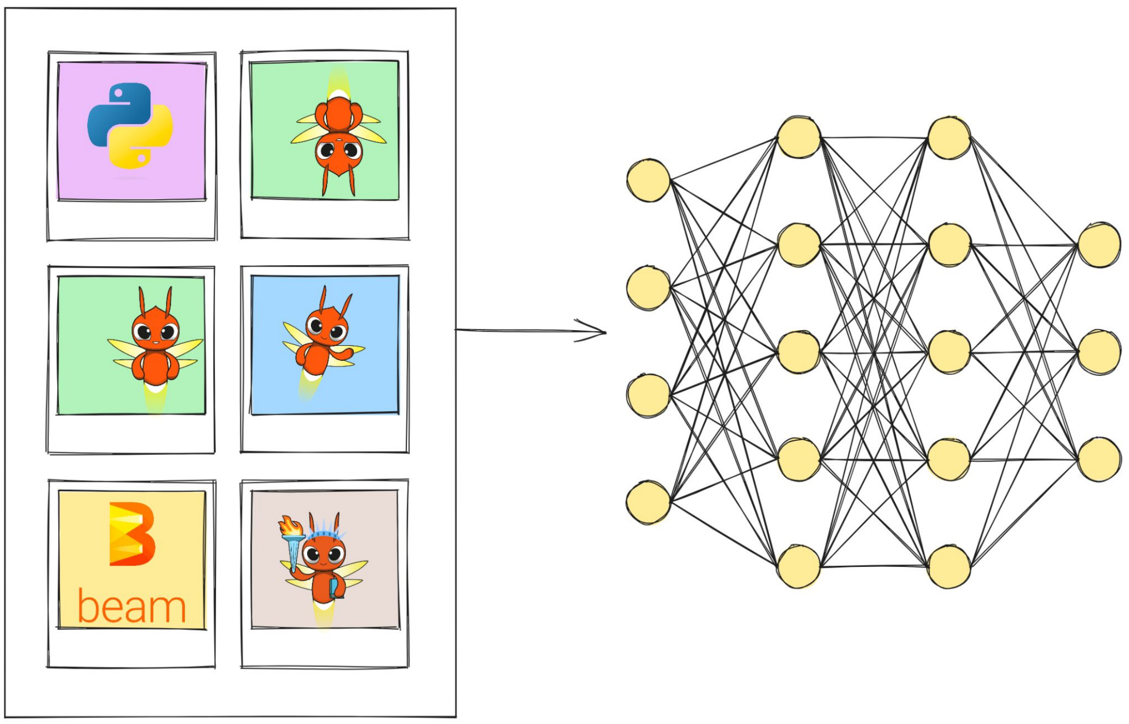
# What is machine learning model training?

```
def contains_firefly():
    ...
```

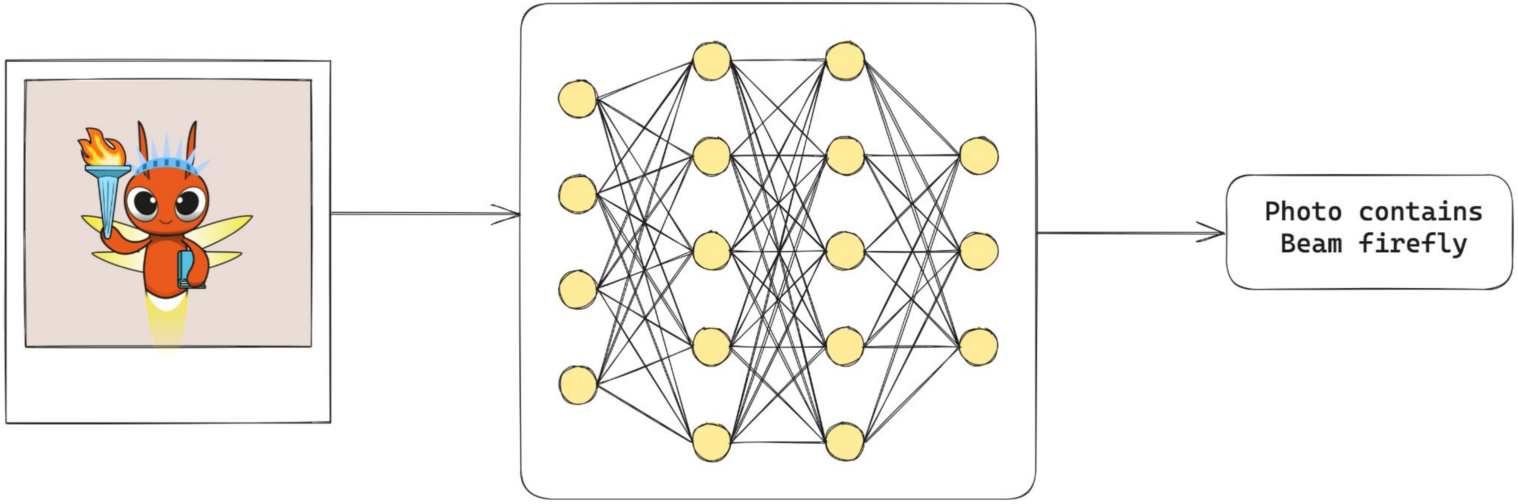



Writing logic to detect the Beam macot is almost impossible

# What is training a machine learning model?

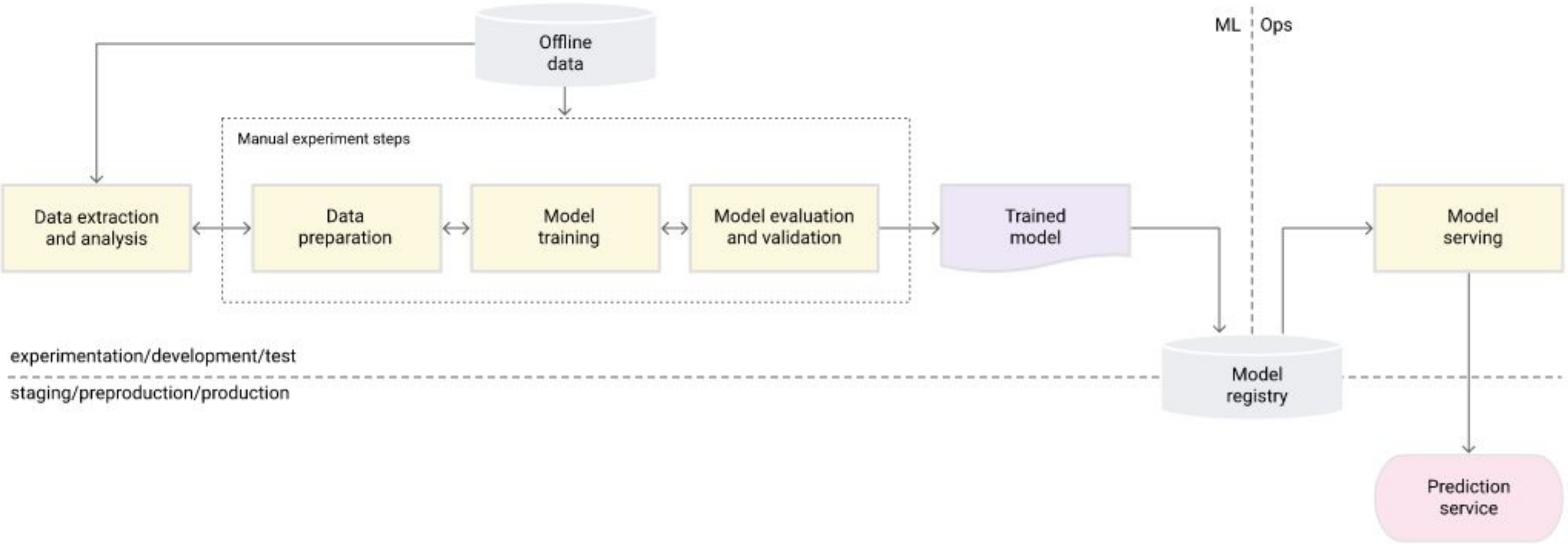


# What is training a machine learning model?

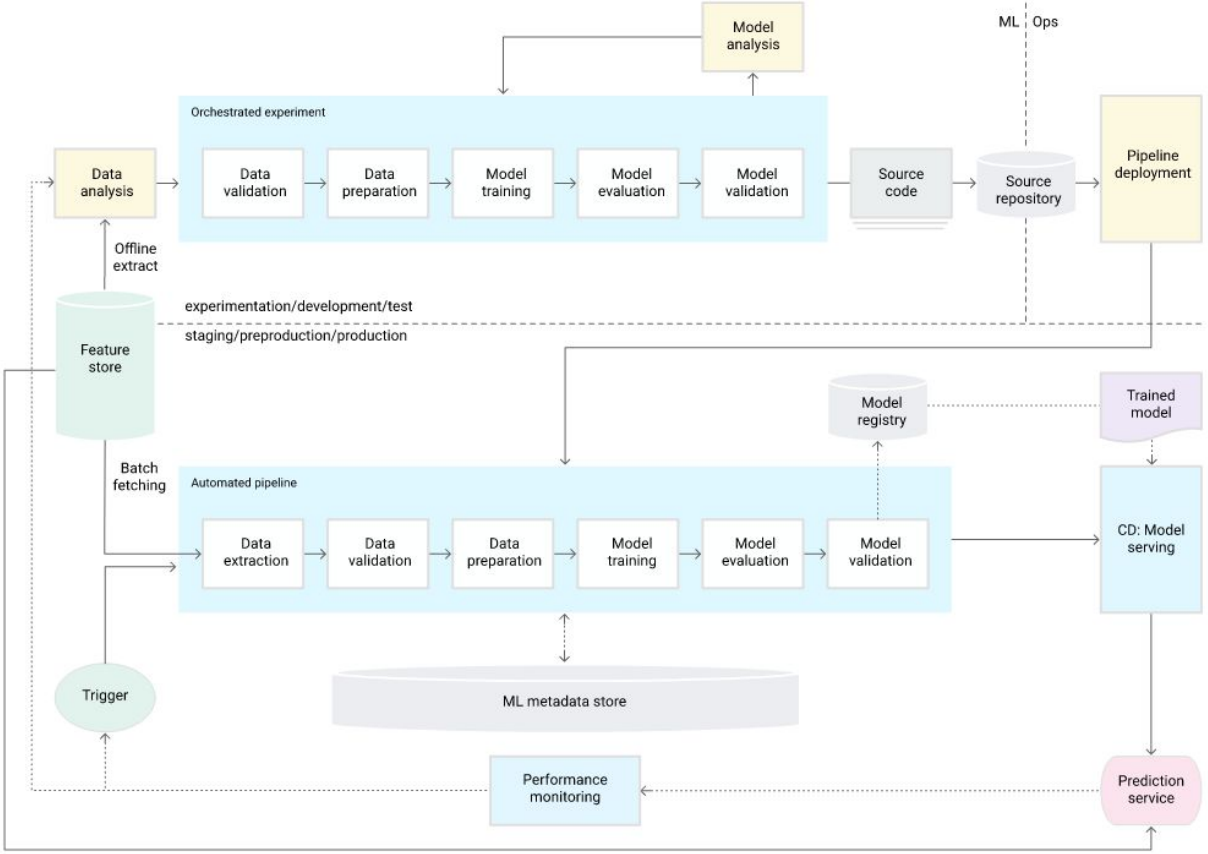


How are machine learning applications built and deployed?

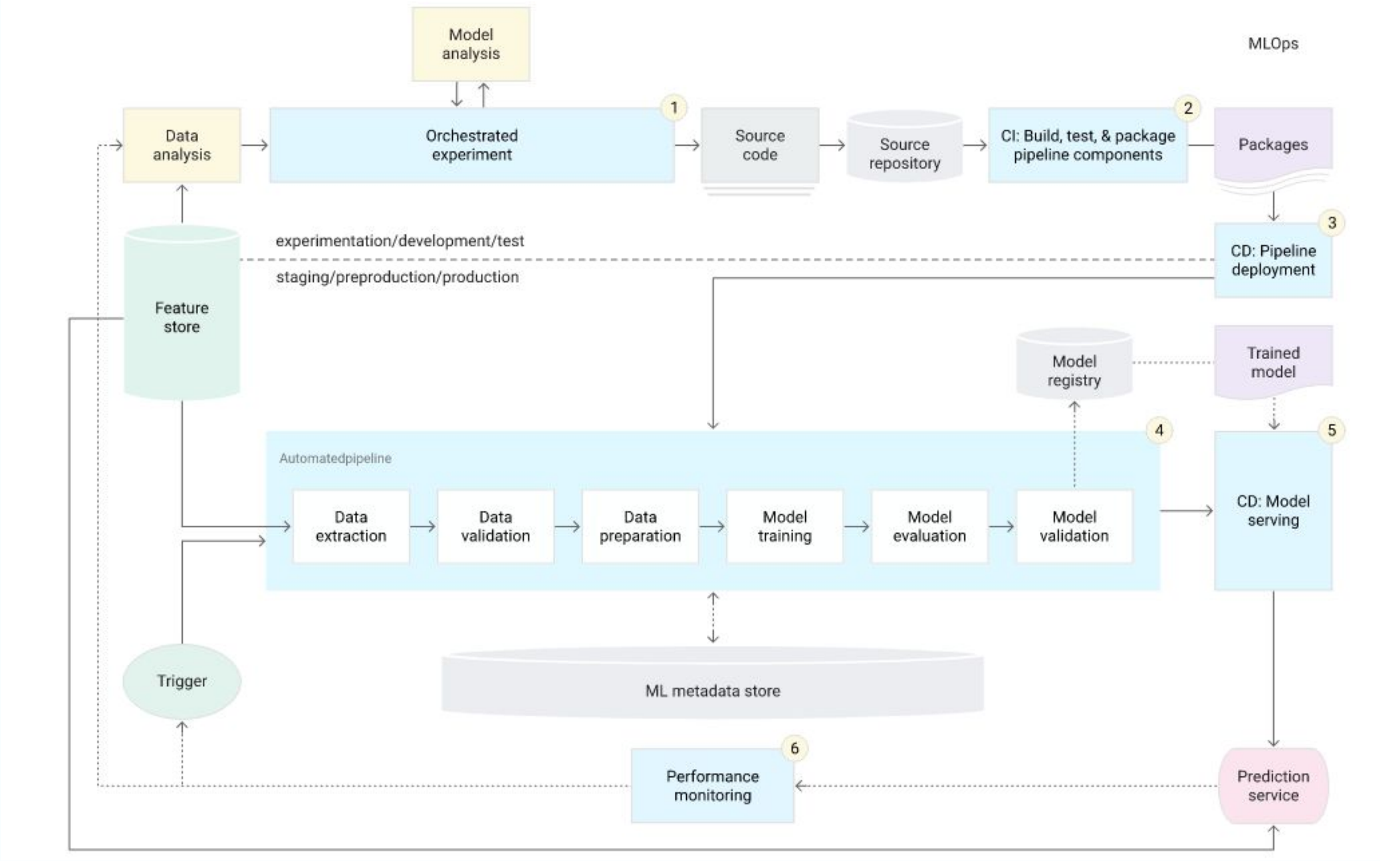
# MLOps: Level 0



# MLOps: Level 1

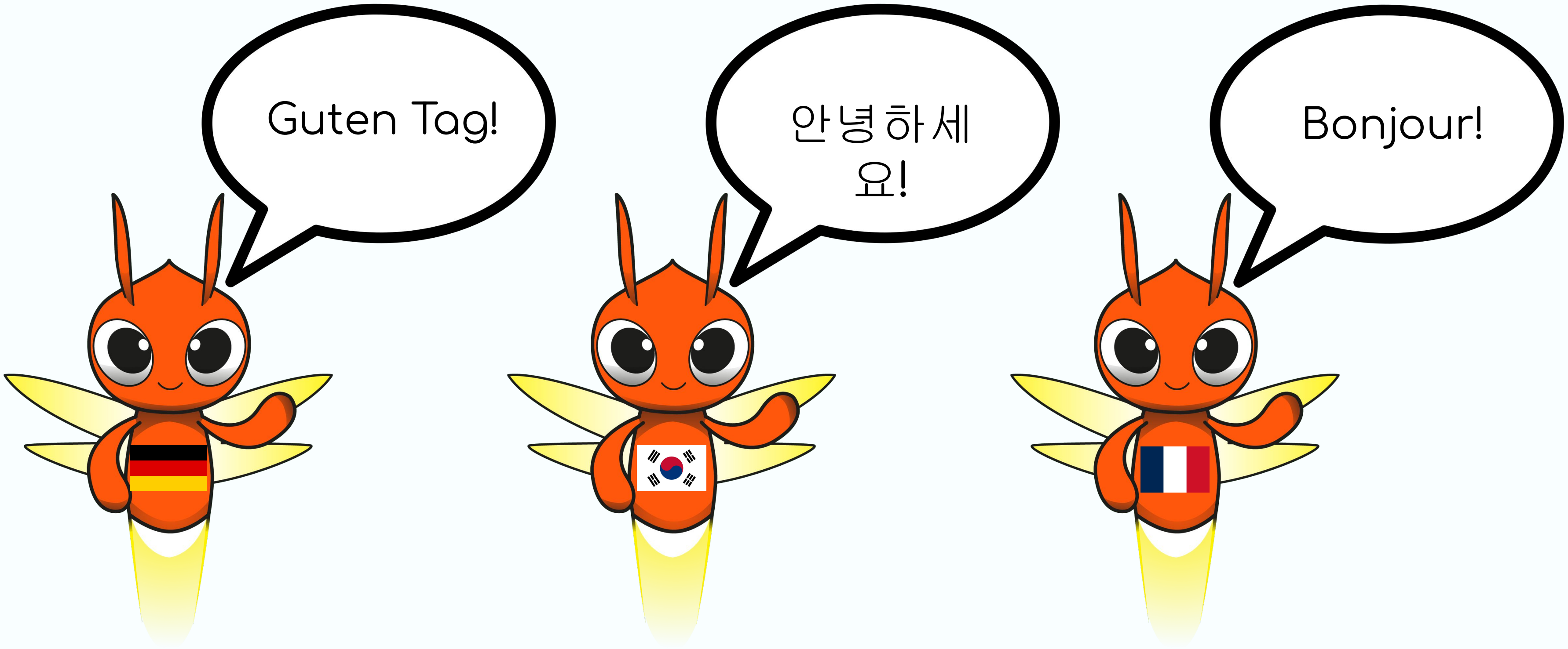


# MLOps: Level 2

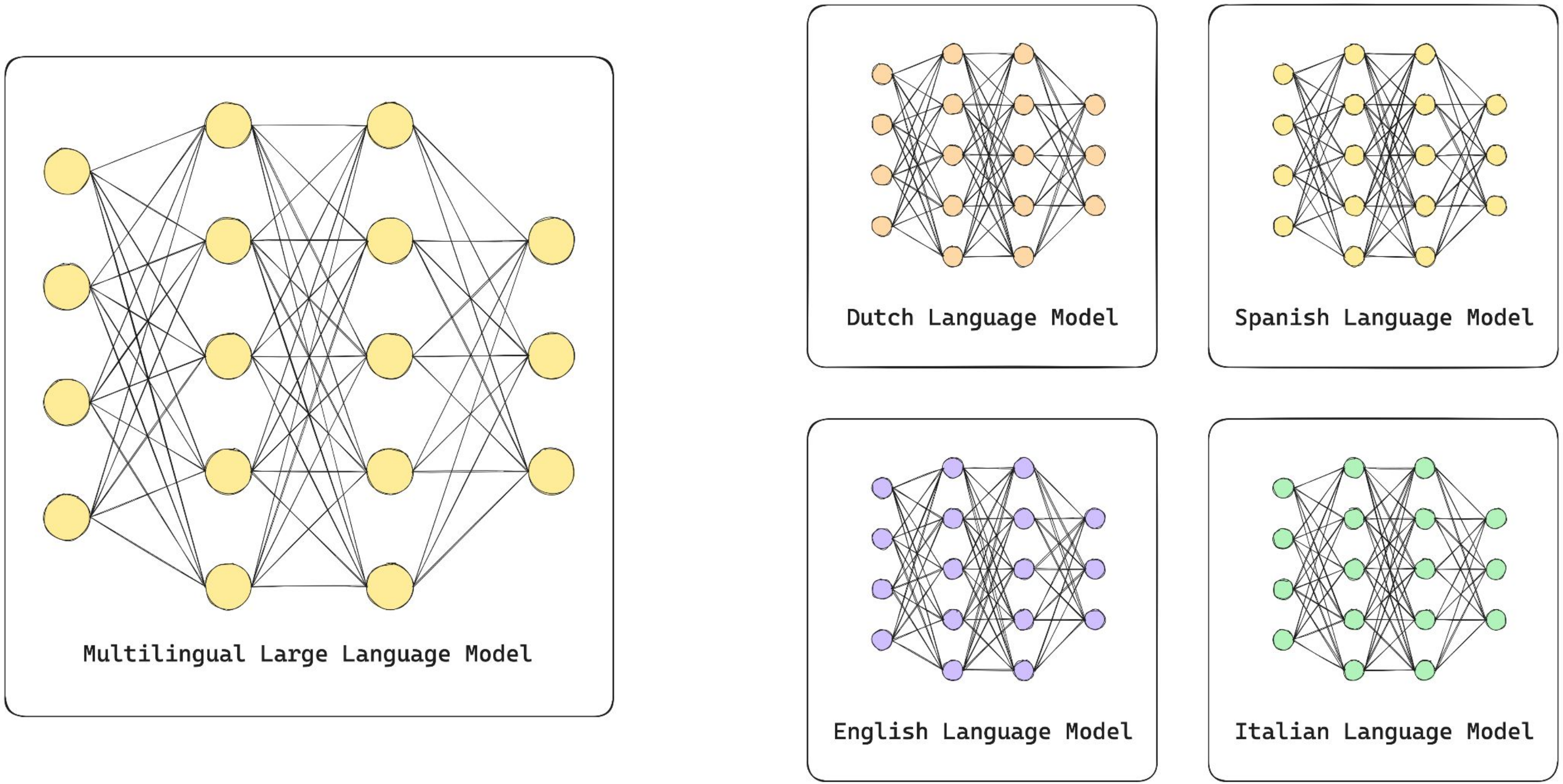


# What is per entity training?

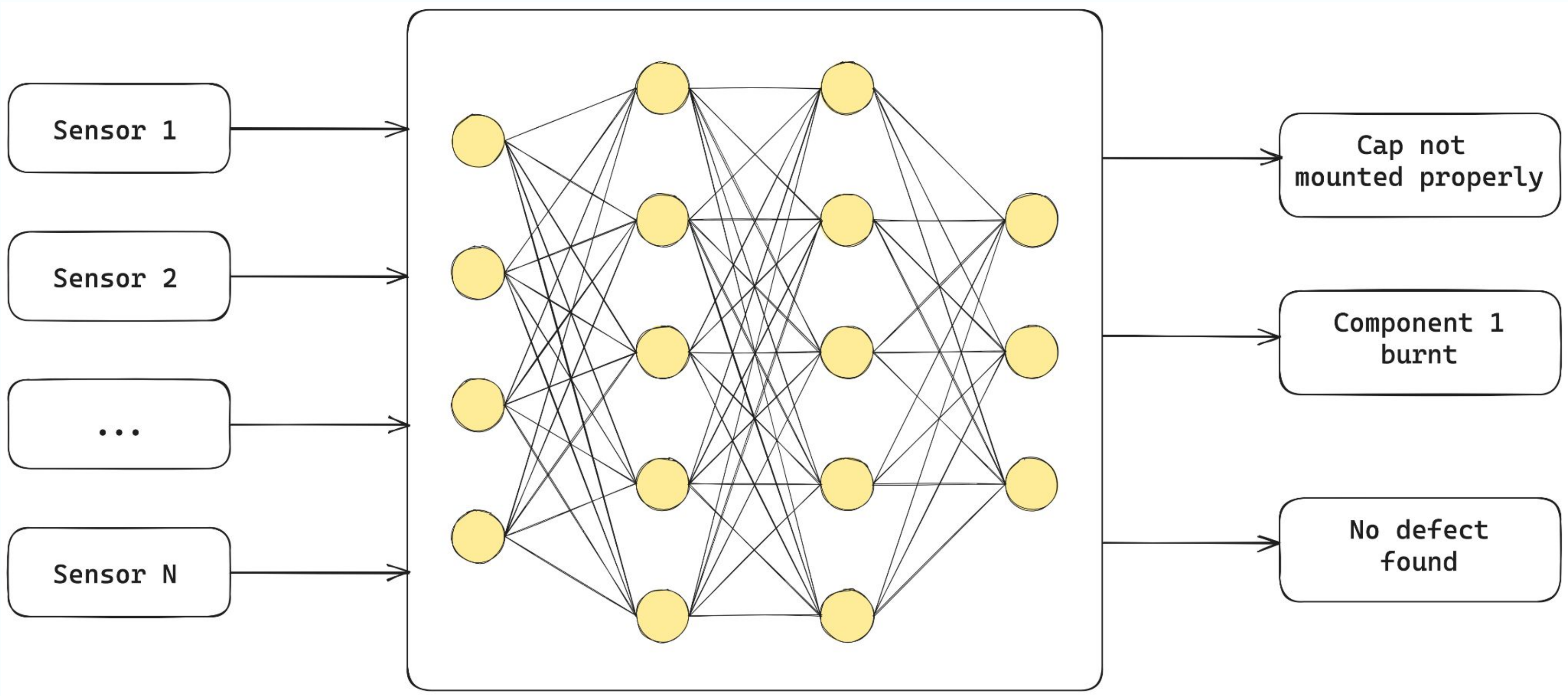
# Example: Building multilingual chatbot



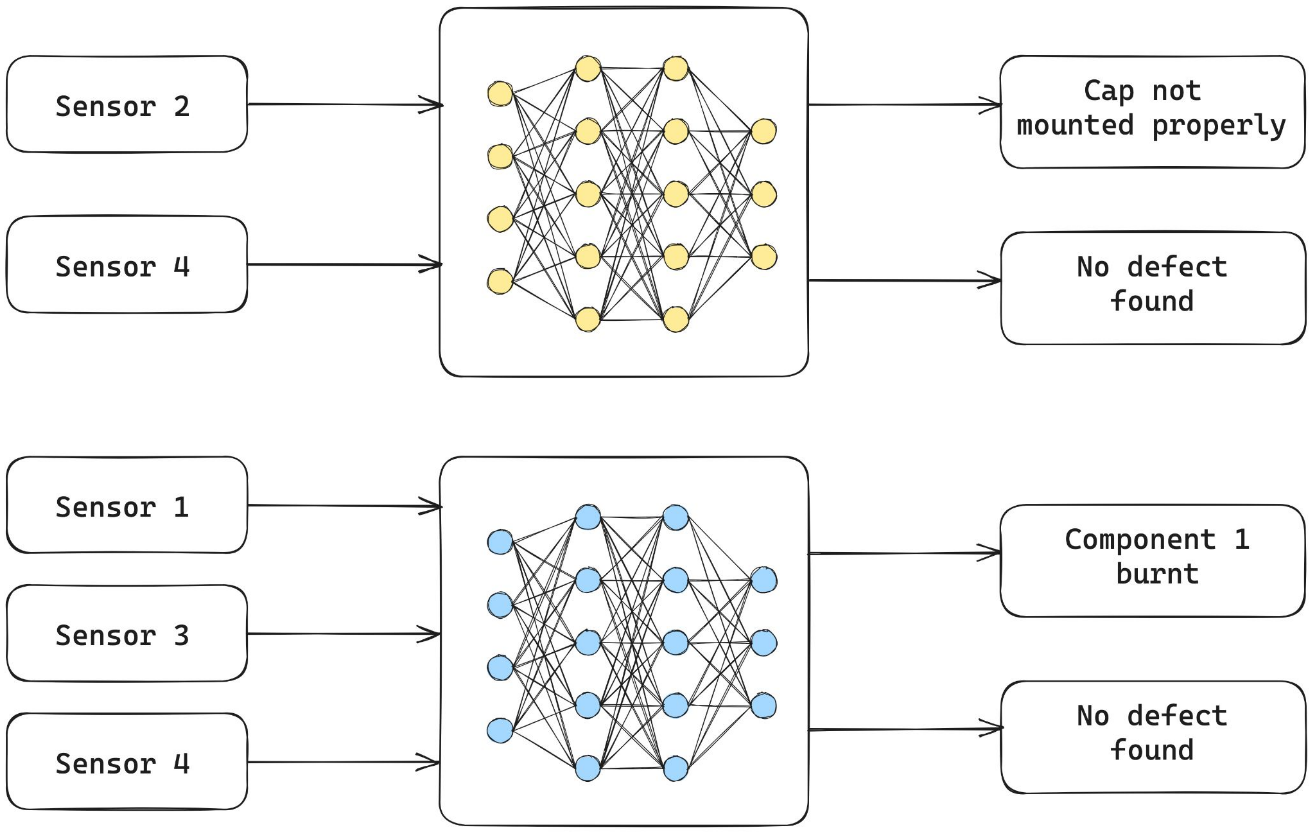
# What is per entity training?



# Example: Detect production defects using sensor data

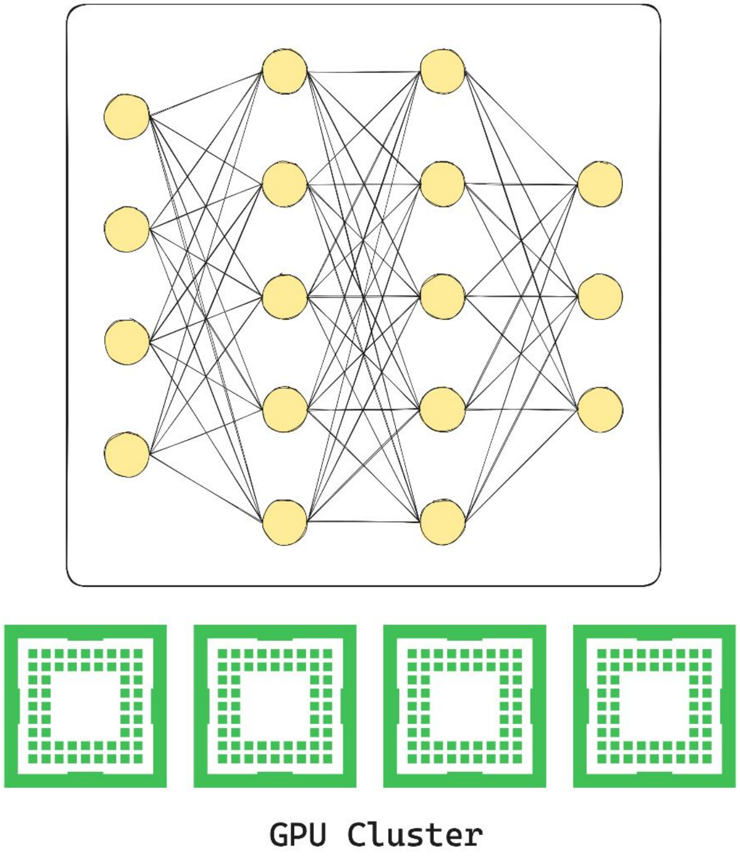


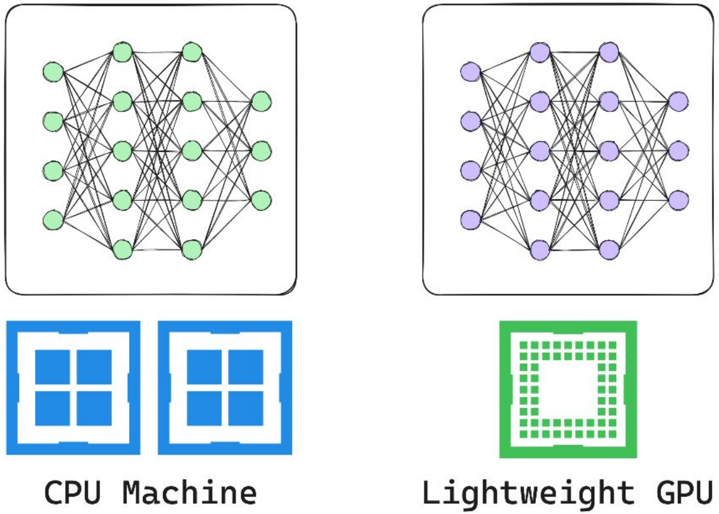
# Example: Detect production defects using sensor data



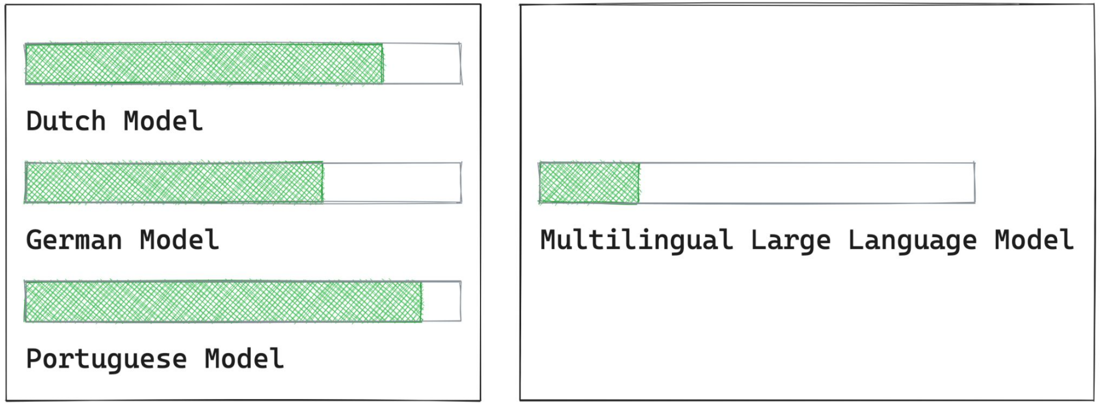
# Why use a per entity strategy?

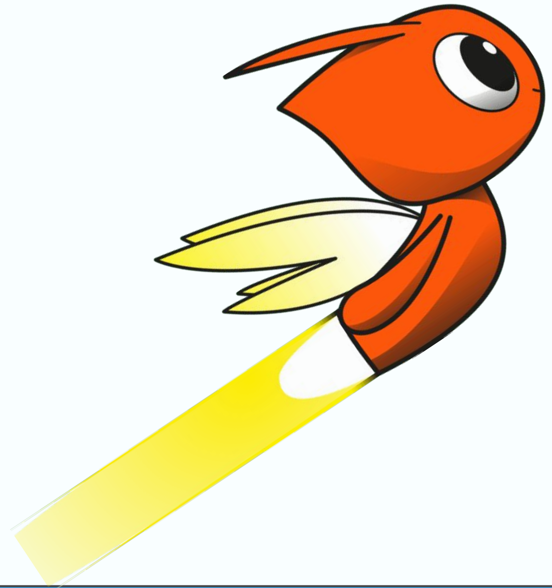
# Reduce Model Infrastructure Requirements



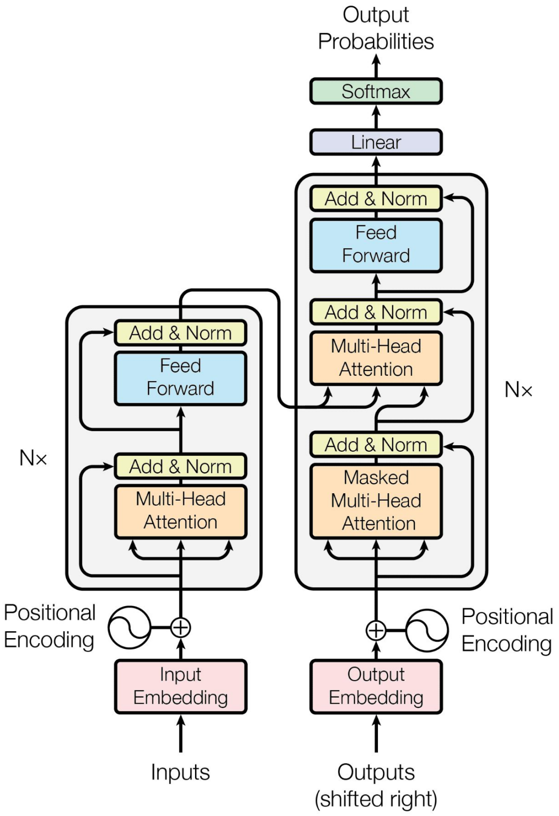



# Faster training & inference

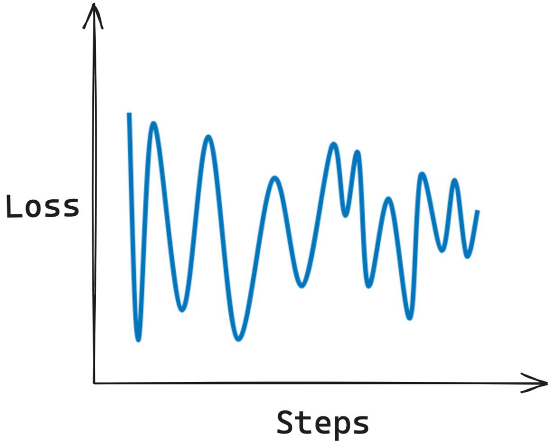


# Address fairness and bias



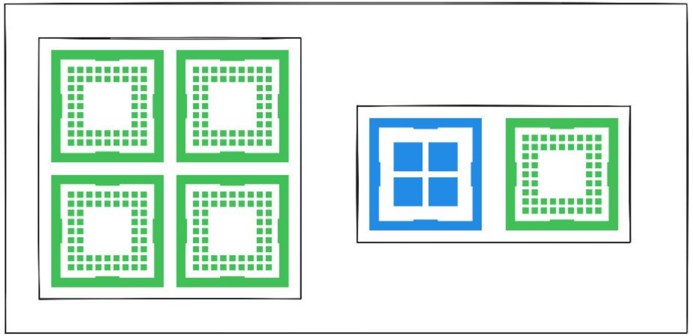

# Easier to detect problems



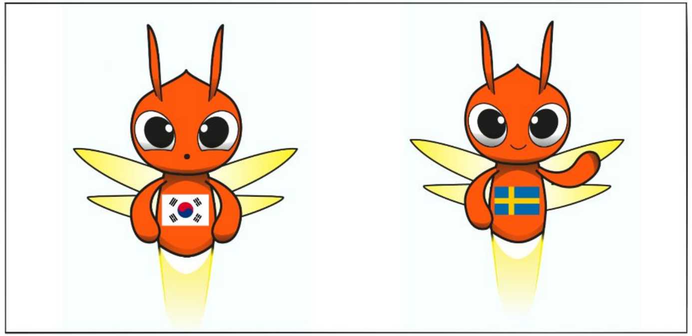

| | | | |
|---|---|---|---|
| .25 | .14 | .36 | .25 |
| .35 | .45 | .08 | .12 |
| .12 | .23 | .33 | .32 |
| .28 | .18 | .23 | .31 |

Confusion Matrix

# Simpler models have the following advantages

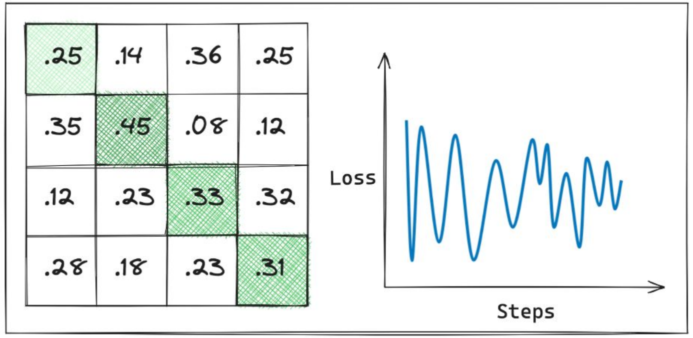


Less powerful hardware required



Easier to address bias

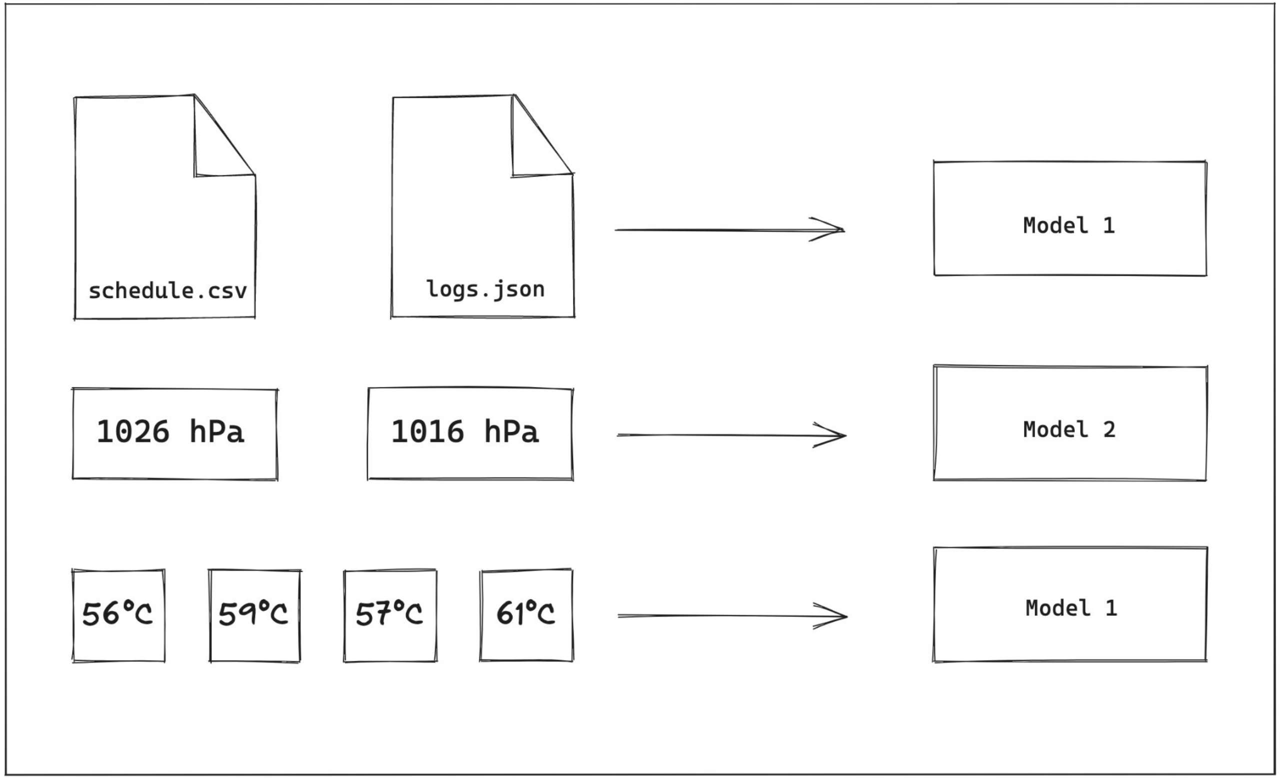

Faster training & inference




Easier debugging

But there is one big problem:
*How do I manage the training of all of these models?*

# Manage training pipelines



# The solution? Apache Beam!

- Apache Beam can handle ***streaming*** and ***batch data***
- Apache Beam can easily ***prepare data*** for training
- Apache Beam can run on different ***runners*** depending on the model's ***requirements***
- ***Abstraction*** in ML libraries allows us to train models with few lines of code

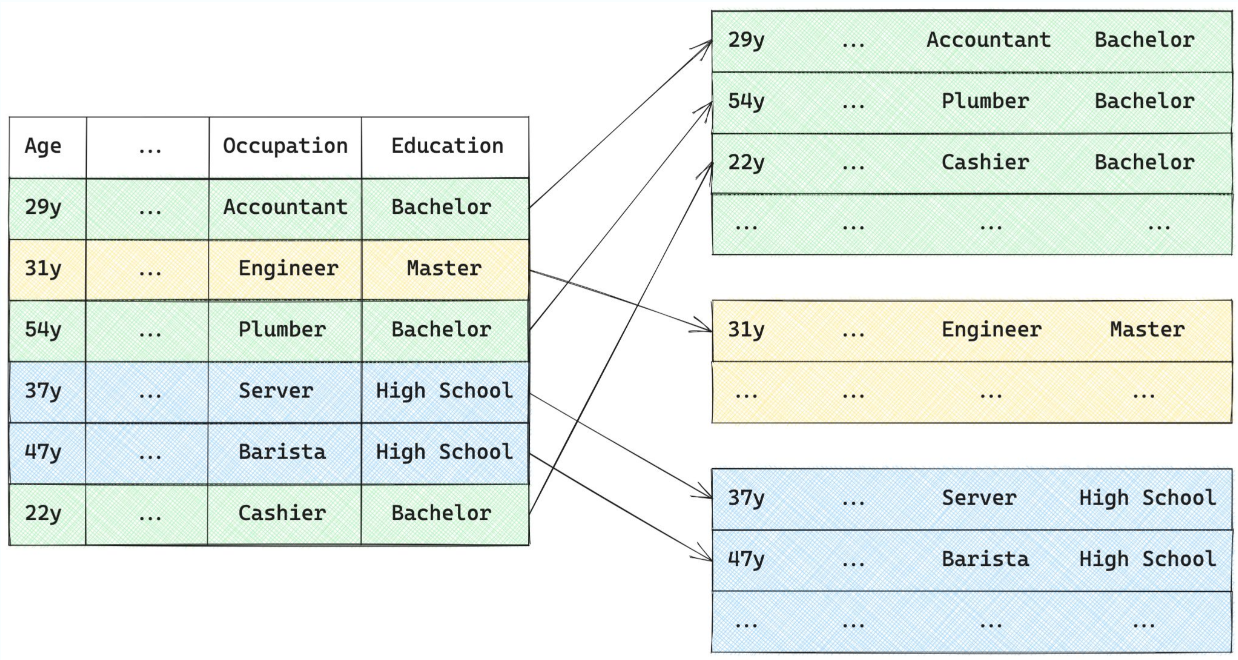
Let's look at an example of a per entity training pipeline

# Predicting incomes per education level

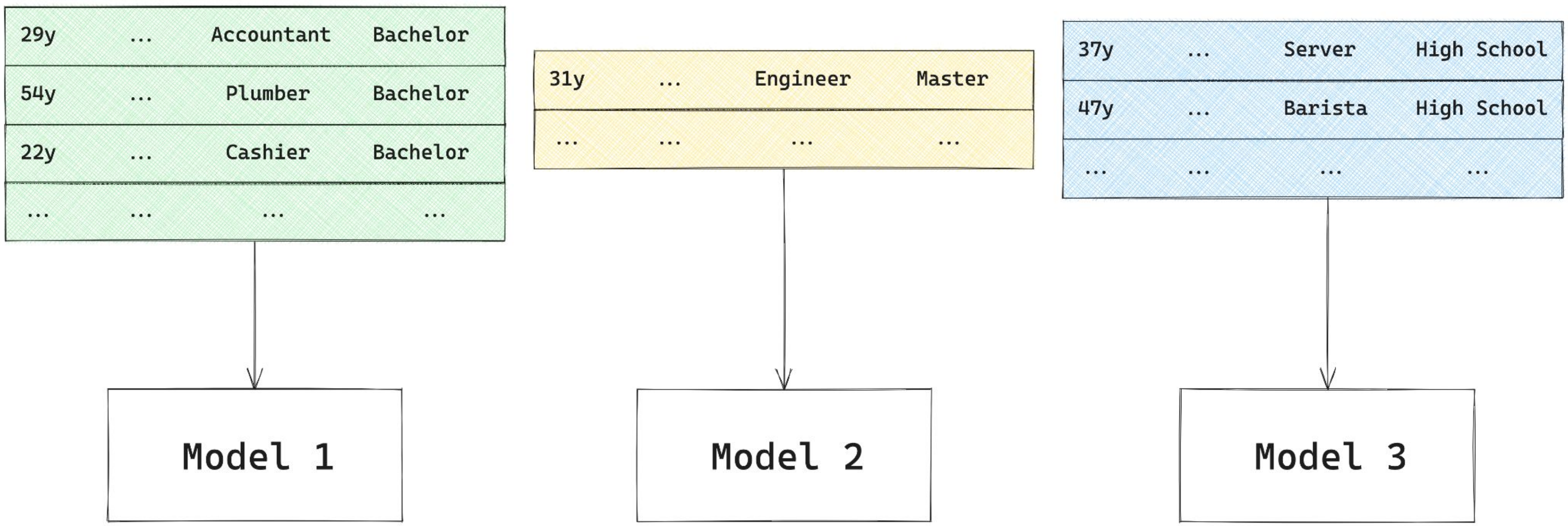
| Age | Workclass | Education | Marital Status | Occupation | Relationship | Race | Sex | Hours per Week | Native Country | Compensation |
|---|---|---|---|---|---|---|---|---|---|---|
| 25 | Private | 11th | Never-married | Machine-op-inspct | Own-child | Black | Male | 40 | USA | <=50K. |
| 38 | Private | HS-grad | Married-civ-spouse | Farming-fishing | Husband | White | Male | 50 | USA | <=50K. |
| 28 | Local-gov | Assoc-acdm | Married-civ-spouse | Protective-serv | Husband | White | Male | 40 | USA | >50K. |
| 44 | Private | Some-college | Married-civ-spouse | Machine-op-inspct | Husband | Black | Male | 40 | USA | >50K. |
| 18 | ? | Some-college | Never-married | ? | Own-child | White | Female | 30 | USA | <=50K. |

# Pipeline overview


# Split data per education level



# Train model per dataset



# Pipeline overview

```
with beam.Pipeline(options=pipeline_options) as pipeline:
    _ = (
        pipeline | "Read Data" >> beam.io.ReadFromText(known_args.input)
        | "Split data to make List" >> beam.Map(lambda x: x.split(','))
        | "Filter rows" >> beam.Filter(custom_filter)
        | "Create Key" >> beam.ParDo(CreateKey())
        | "Group by education" >> beam.GroupByKey()
        | "Prepare Data" >> beam.ParDo(PrepareDataforTraining())
        | "Train Model" >> beam.ParDo(TrainModel())
        | "Save" >> fileio.WriteToFiles(path=known_args.output,
sink=ModelSink()))
```

# Step 1: Data preparation

```python
def custom_filter(element):
  return len(element) == 15 and '?' not in element \
      and ' Bachelors' in element or ' Masters' in element \
      or ' Doctorate' in element
```

# Step 1: Data preparation

```python
class PrepareDataforTraining(beam.DoFn):
  def process(self, element, *args, **kwargs):
    key, values = element

    #Convert to dataframe
    df = pd.DataFrame(values)
    last_ix = len(df.columns) - 1
    X, y = df.drop(last_ix, axis=1), df[last_ix]

    # select categorical and numerical features
    cat_ix = X.select_dtypes(include=['object', 'bool']).columns
    num_ix = X.select_dtypes(include=['int64', 'float64']).columns

    # label encode the target variable to have the classes 0 and 1
    y = LabelEncoder().fit_transform(y)

    yield (X, y, cat_ix, num_ix, key)
```

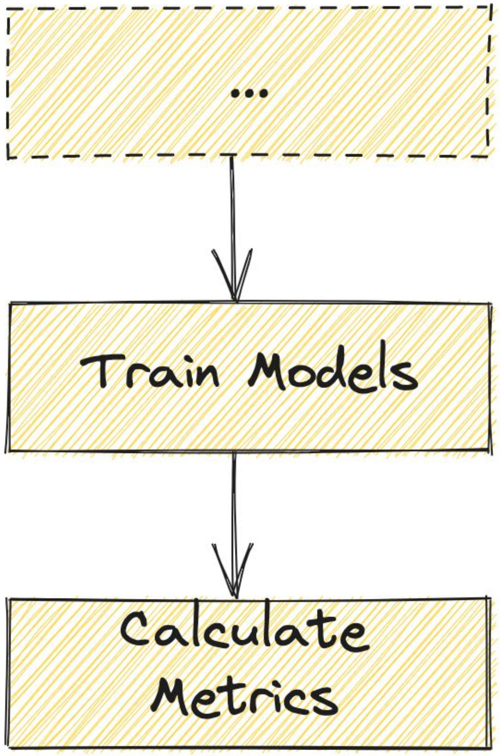
# Step 2: Training the models

```
class TrainModel(beam.DoFn):

  def process(self, element, *args, **kwargs):
    X, y, cat_ix, num_ix, key = element
    steps = [('c', OneHotEncoder(handle_unknown='ignore'), cat_ix),
             ('n', MinMaxScaler(), num_ix)]

    # one hot encode categorical, normalize numerical
    ct = ColumnTransformer(steps)

    # wrap the model in a pipeline
    pipeline = Pipeline(steps=[('t', ct), ('m', DecisionTreeClassifier())])
    pipeline.fit(X, y)

    yield (key, pipeline)
```

# Step 3: Saving models

```
class ModelSink(fileio.FileSink):
  def open(self, fh):
    self._fh = fh

  def write(self, record):
    _, trained_model = record
    pickled_model = pickle.dumps(trained_model)
    self._fh.write(pickled_model)

  def flush(self):
    self._fh.flush()
```

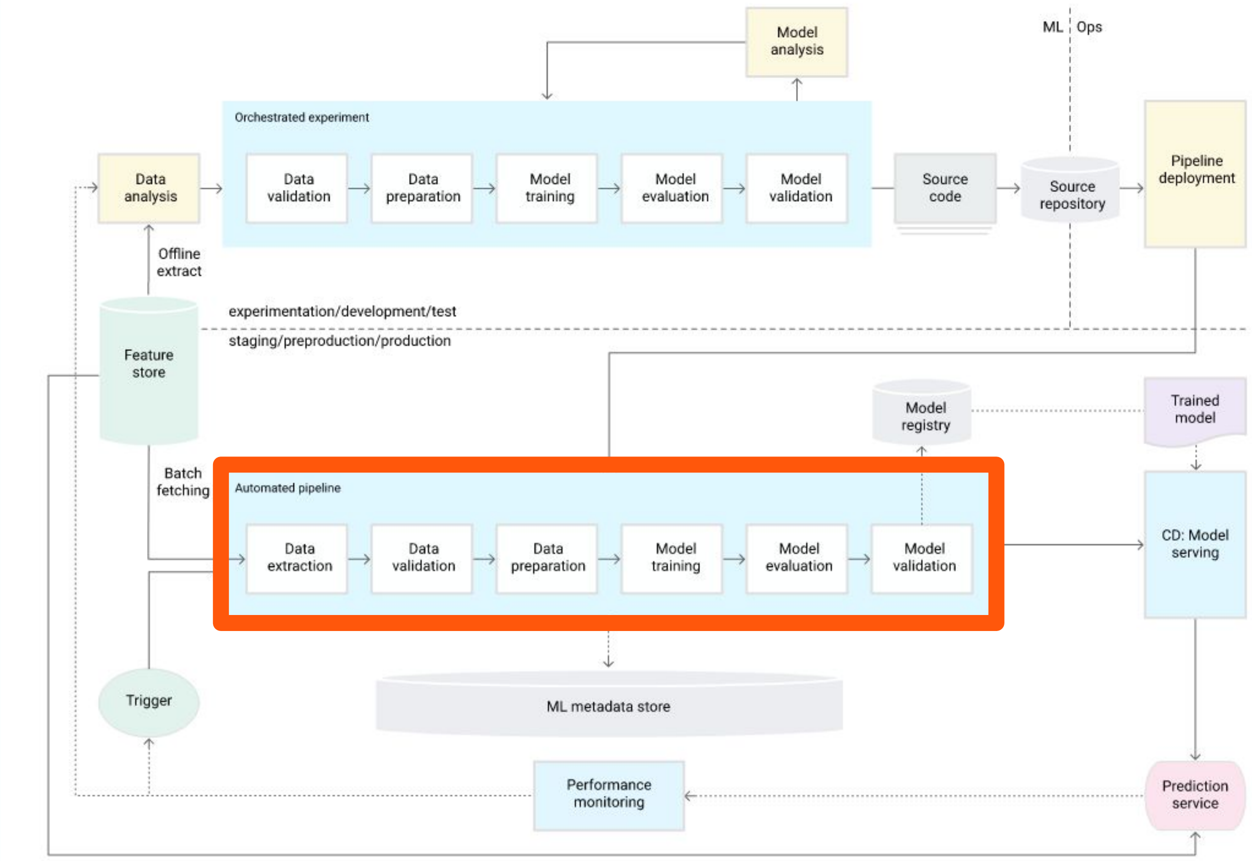
# Extending the pipeline



# Extending pipeline with metrics

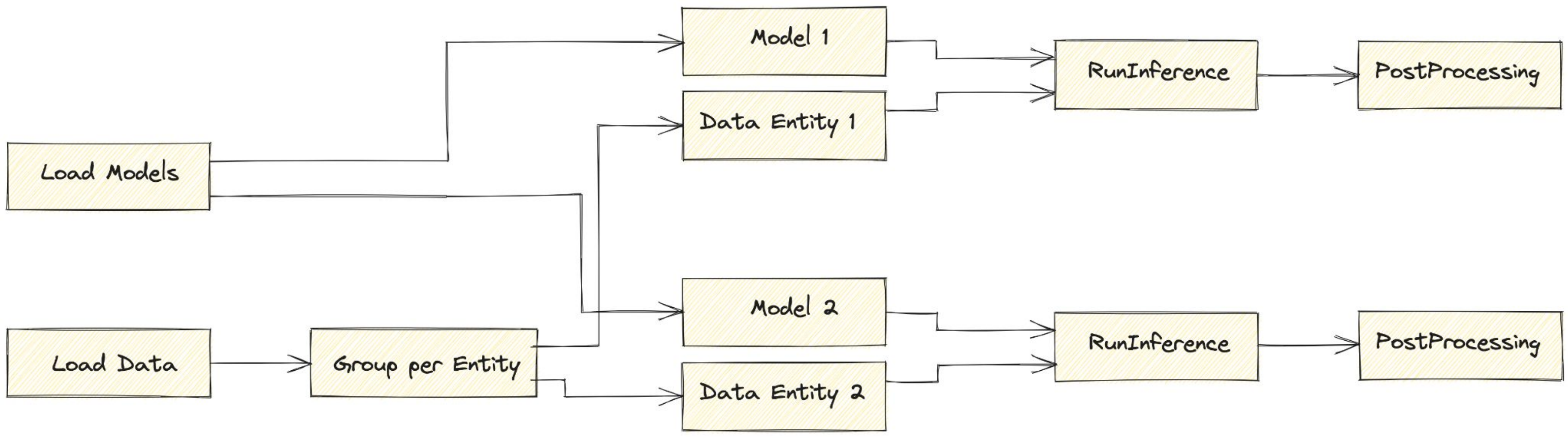
```
class EvaluateModel(beam.DoFn):
  def __init__(self, model_uri):
    file = FileSystems.open(model_uri, 'rb')
    self.model = pickle.load(file)

  def process(self, element, *args, **kwargs):
    inputs, labels = element
    predictions = self.model.predict(inputs)
    accuracy = sklearn.metrics.accuracy_score(y_pred=predictions,
y_true=labels)
    f1 = sklearn.metrics.f1_score(y_pred=predictions, y_true=labels)
    recall = sklearn.metrics.recall_score(y_pred=predictions, y_true=labels)

    file = FileSystems.open(f'model_uri_metrics', 'web')
    file.writelines([f'accuracy: {accuracy}', f'f1: {f1}', f'recall:
{recall}'])
```

# How does this pipeline fit in the MLOps architecture?



Let’s try out our model using the RunInference trasform

# Bonus: Inference in Apache Beam



# Summary

- Apache Beam is more and more becoming technology that can be used in advanced MLOps setups
- Per entity strategy has several advantages
  - Requires less powerful hardware
  - Faster training and inference
  - Easier to address bias
  - Easier to debug
- Apache Beam a perfect candidate for per entity training pipelines thanks to
  - Excellent for data preprocessing and preparation
  - Different runners depending on model requirements
  - Abstraction in ML libraries that make it easy to train a model

Jasper Van den Bossche

# QUESTIONS?

https://www.linkedin.com/in/jasper-van-den-bossche/
https://github.com/jaxpr
https://www.ml6.eu/