Implementation of a Predictive AI to Feed Simulation

Carlo Simon, Merlin Hladik, and Natan Georgievic Badurasvili
Hochschule Worms
Erenburgerstr. 19, 67549 Worms, Germany
e-mail: {simon,hladik,natan.badurasvili}@hs-worms.de

***Abstract*—This paper extends a former version presented at the SIMUL 2024 conference. Its topic resulted from a remark of an industry partner dissatisfied with the result of a traffic simulation of a warehouse. Although the simulation was a replica of the behavior on base of the current order data, it deviated from the real situation since many distributors do not adhere to their orders. Methods of predictive artificial intelligence and especially machine learning have been identified to adapt the simulation input on the base of past schedules. The paper answers the question on how to extend the previous simulation model by a suitable forecast component and explains the implementation in more detail than the original paper. Unfortunately, the industry partner does not collect the data needed for such a forecast. Therefore, test data was generated which is explained, too. By the example of a real-world warehouse scenario, the simulation of its traffic and the information needed for this is demonstrated. Afterwards, the necessary extensions of the data set are explained and how to set up a machine learning component to predict future deviations of schedules. The adapted schedules can then be simulated to create alternative schedules. Like in a construction manual, the implementation is explained step by step.**

***Keywords*-Predictive Artificial Intelligence; Neural Networks; Machine Learning; Simulation; Petri Nets.**

I. Introduction

The idea to use AI to feed a simulation [1] was initiated by discussions during the SIMUL 2023 conference [2]. The participants had various imaginations about the impact of currently discussed Artificial Intelligence (AI) methods like transformers on simulation. While simulation is about causality, many AI methods are about correlation. The participants generally doubted that AI will substitute current simulation methods. However, there is still the possibility to pair both approaches.

This discussion occurred at the right time, as there was a need to address a problem that had arisen in an industry project using alternative methods. They could simulate the incoming and outgoing traffic of a large warehouse for registered transports as described in Section II with the aid of a high-level Petri net. But this did not take into account that the registered (planned) arrival time and the actual arrival time of the transports often do not match.

An investigation began into possible reasons for late transports, such as transport distance, the shipping agent, the producer, or current weather conditions. None of these factors were considered during the actual simulation of the warehouse's inbound and outbound traffic.

Unfortunately, the industry partner does not collect information about delays and, hence, cannot provide empirical values. Nonetheless, there was a desire to further develop the simulation environment by incorporating AI technologies. This idea led to the research question:

> *How can we extend our simulation model by a forecast component based on machine learning?*

A demonstration of the feasibility of this approach has been presented at the SIMUL 2024 conference [1]. The extension within this paper explains the implementation in detail.

Before explaining the new AI specific components of the simulation environment, the paper continues in Section II with a description of the high-level Petri net simulation model for the inbound and outbound traffic of a warehouse. Particular attention is given to the data required for simulation. Afterwards, in Section III, a brief introduction is given to the necessary Machine Learning (ML) methods. Both topics are combined in Section IV to define the structure of a possible ML data set for the problem first and to explain the ML approach in Section V next. The implementation of the ML component is explained in three steps in Sections VI, VII, and VIII. Section IX demonstrates how to integrate the ML approach into the existing environment. The paper ends in Section X with a conclusion and an outlook on future work.

II. Real-World Problem

The industry partner at an industrial park (Industriepark Höchst, *ISL*) provides logistics services for chemical, pharmaceutical and healthcare industries and currently expands its logistics services. Freely published information show the size of this venture [3]:

- Space for more than 21,000 pallets
- 9 separate warehouse sections
- Storage of multiple hazardous material storage classes for chemicals and pharmaceutics
- A wide temperature range from -6 to 20 degrees Celsius in the different sections of the warehouse

As depicted in Figure 1, the warehouse can be accessed via ramps (2). Each section has a loading zone (3) and the actual storage zone (6).

During planning and go live, the inbound and outbound traffic of this warehouse has to be simulated to objectify assumptions made during the conceptual phase. A typical inbound process is conducted as follows: before approaching the industrial park, the shipping agent books a time slot in advance. When trucks arrive, the drivers register with

the gatekeeper in the office (1). Afterwards, they dock at the ramp they have been assigned to (2). Then, the goods are picked by standard forklifts (called VHS) and placed in the staging zone (3). After the truck has left, the goods are carried through the driving zone (4) and dropped on a handover point (5). Finally, narrow aisle forklifts (called SGS) pick the goods and store them in the high rack storage area (6).

Outbound processes are executed in reverse order, except that the goods are provided in the staging zone before trucks arrive.

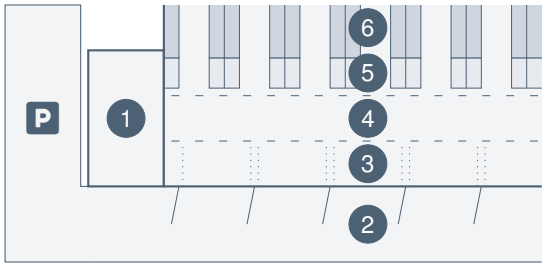

Figure 1. Sketch of the Warehouse.

Various specifics may be excluded from the simulation, such as the commissioning of certain goods. Furthermore, the precise positioning of goods within the warehouse, and consequently the exact driving times, hold diminished importance. In lieu of this, reasonable average times have been selected to replicate typical operational norms.

Model and simulation have been described in detail in [2] and [4]. The model consists of state machines for the inbound and outbound processes and a high-level Petri net to execute these state machines in parallel. Also, time constraints and restricting resources are taken into account.

Two conceptual data models for orders and resources are needed for this. The resources are discussed, first. Table I shows the data needed for simulation based on one kind of resource. This information is spread over several tables (or, in terms of Petri nets, over several places) for each kind of resource.

TABLE I. Attributes for the Resource Places

Attribute	Description
id	id for resources of this kind
product	product group
free	available or locked
timestamp	timestamp of the latest state change
order	assigned order id

Attribute *id* may identify a specific resource like a numbered ramp or a dedicated resource of this kind, like one of the VHS. *product* describes the resource allocation to chemical or pharmaceutical. *order* references to the order that uses this resource and simplifies joins among the different data sets.

Table II shows the data needed for orders. Beside an identifier *id* and the specification whether the order belongs to a chemical or pharmaceutical *product*, the *total* number of pallets for the transport is specified. *status* identifies the current order's state and, implicitly, whether this is an inbound or outbound process.

TABLE II. Attributes of an Order's State

Attribute	Description
id	order id
product	product group
total	total amount of pallets requested
status	initial or current order status
ramp	target ramp
arrival	scheduled time of arrival
preparation	scheduled time of completed staging
fillHandover	amount of pallets in handover areas
fillRamp	amount of pallets at target ramp
fillTruck	amount of pallets in truck
usedGate	used resource *gate*
usedSGS	used resource *SGS*
usedVHS	used resource *VHS*
timestamp	timestamp of the latest state change

One or two times must be defined per order: for both inbound and outbound, the *arrival* time of the truck is given due to its registration. Outbound processes additionally need a *preparation* time when staging begins. This staging time leaves room for optimisation for the warehouse operators.

At the end of the simulation explained in the following, all changes to orders are exported to a dashboard. A *timestamp* traces the moments these changes occur. To simplify the computing of this visualisation, the allocated resources like *ramp*, *usedGate*, *usedSGS*, *usedVHS* are saved. Finally, the amount of goods at the different places is stored in the attributes *fillHandover*, *fillRamp*, and *fillTruck*.

Without having an impact on the result, the real process and the simulated one differ slightly. For the simulation, the ramps are assigned in advance; in real world the gatekeeper decides on the ramp based on personal experience.

The *Process-Simulation.Center (P-S.C)* was chosen for modelling and simulation [5]. Since the *P-S.C* is a Petri net based Integrated Management System, it fulfils further constraints important for the industry partner among the pure ability to simulate. Safety and security aspects are of high priority to ISL. Therefore, access to and visibility of (real-world) data must be limited.

The *P-S.C-Cloud* (the *P-S.C* is only provided via internet) and the multi-client capability of the *P-S.C* help to manage security issues [5]: The tool itself only runs locally in a browser with data never leaving the system. Sensitive input data and simulation results can be stored on in-house servers without the *P-S.C-Cloud* ever coming in contact.

In addition, the industry partner requires a user interface (UI) to edit the simulation parameters (orders, times, resources, priorities) easily. The simulation results ought to be presented in a descriptive dashboard. Today, such an

interface is called a digital shadow, a piece of software which maps real-world data and processes to a virtual world [6].

Figure 2 explains the interaction between the *P-S.C* and the local UI using CSV-files. Simulation parameters can be imported this way, too.

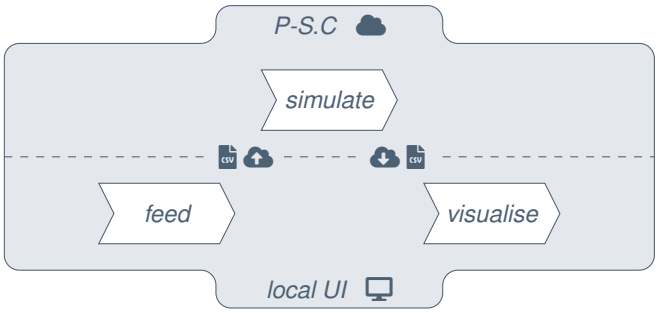

Figure 2. Coupling of Digital Shadow (local UI) and *P-S.C*.

The execution of a specific simulation scenario is divided into three distinct phases:

- **feed:** The local UI is implemented as a web-based application that can be used with any internet connected computer or iPad. It is used to enter the simulation parameter, priorities and especially the actual order data.
- **simulate:** The *P-S.C* loads the data into the browser of the end user to start the simulation. The *P-S.C-Cloud* does not get in touch with it which guarantees full control over the data. After the simulation has finished, an automatic download is started and the users can store the results on local hardware.
- **visualise:** The downloaded file in turn can now be uploaded in a dashboard which is also implemented in the local UI. This helps the end-users to find the best strategy for driving the warehouse. In particular, workload spikes, transportation bottlenecks, and staff occupancy can be analysed and visualised.

This approach is limited to simulate one specific schedule of orders for one specific period, e.g., one day. It is not flexible concerning deviations of the transports. The next sections explain how to extend this environment to handle this limitation.

III. Machine Learning Fundamentals

Artificial Intelligence (AI) is defined in many different ways. The most common definition of AI is that of a rational agent. Such an agent tries to provide the best (expected) outcome, given its inputs. What constitutes the best outcome remains a matter of definition [7].

If a rational agent is created to improve the provided output by gaining some sort of experience, this is called Machine Learning (ML). Figure 3 shows the idea of ML as described by [8]. The figure depicts the rational agent as a model. Its task (red) is to compute an output when presented with input data. The model obtains a mapping based on training data. Its formation is the learning problem (blue). Which algorithms are used depends on the chosen ML model [8].

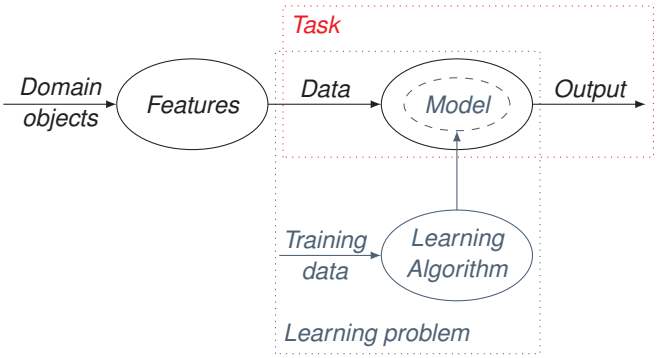

Figure 3. Machine Learning as explained by [8].

One possible ML model is Deep Learning (DL). A DL model is an artificial neural network with several (hidden) layers. A neural network is to mimic the principle of real neurons. Single neurons are connected by directed, weighted arcs. If the incoming arcs yield a high enough activation potential, the activation function in a neuron triggers a corresponding output. The weighs constitute the main parameters of this ML model type [9].

If the training set includes results, this is called supervised learning. In this case, the data is considered labelled. The metric used to examine the model's quality is to compare the actual results with the computed ones [10].

Two kinds of problems may be solved with the aid of artificial neural networks:

- **Classification**: In mathematical notation, a classifier is a function $y = f(x)$, where x is the input data item and y is the output category [11].
  Applied to our example of a warehouse, classification could help to predict which transports may be unpunctual.
- **Regression**: In regression, we try to understand the data points by discovering the curve that might have generated them [11].
  Applied to our example of a warehouse, regression could help to predict how many minutes transports may be unpunctual.

IV. Extended Data Set for ML

The aim of the presented approach is to correct and complete future orders and especially to forecast

- which future transports will or will not be in time, i.e., to solve a classification problem, and
- to forecast the expected deviation of the arrival time of future transports, i.e., to solve a regression problem.

But which parameters may influence the arrival time of trucks? And how can this information be coded, such that it can be processed by a learning algorithm?

First, it is worth to consider the data used for simulation already. All "internal" parameters of the orders like order *id*, current *timestamp* of the simulation, allocated *ramp* and other resources (*fillHandover*, *fillRamp*, *fillTruck*, *usedGate*, *usedSGS*, or *usedVHS*), and the *preparation* time for outgoing orders can be ruled out as possible sources.

The probably most valuable source is the planned *arrival* time coded as a timestamp consisting of date and time. This information, which is typically stored as a string, should be preprocessed for a learning algorithm. The following information can be extracted and coded as discrete numbers:

- The month or season of arrival, which may have an impact due to weather conditions.
- The arrival time in hours or at least in categories like early, in the middle of the day or by the end of the day, which may have an impact on the transportation conditions like traffic or the work schedule of the drivers.
- The day of week which may have an impact on the traffic intensity.

Also, the *state* attribute may be of interest because it is used to differ between incoming and outgoing transports. For outgoing transports, "only" an empty truck has to arrive while incoming transports need to be prepared before they reach the warehouse. This might have an impact on the arrival time.

Attribute *product* is not as valuable as supposed by its name. The simulation only distinguishes between chemical and pharmaceutical products which is very rough. It is not obvious that these two categories correlate with a deviation since there are hundreds of different concrete products that belong to these two groups.

Finally, the *total* amount of pallets is a candidate which might have an impact on the arrival time, especially for incoming transports. Large amounts of goods may cause more problems when being loaded compared to smaller ones.

Further attributes might have an impact which are not considered during simulation. The following attributes have been identified:

- The transportation *distance*, especially if a truck has to arrive from outside of the industrial park or whether it is an internal transport.
- The *shipping agents* may differ concerning their quality standards and the accuracy of delivery time.
- Finally, the producer might have an impact on the time a transport can start after being loaded and, hence, whether it arrives on time.

All remaining values can be enumerated and can hence be used for training of a neuronal net.

Unfortunately, the industry partner does not track these additional information. Even the deviation between the planned and the actual arrival time is not documented. Hence, the following considerations are only of theoretical nature.

V. ML for Predictive Scheduling

In the context discussed in this paper, Deep Neural Networks (DNNs) can be used threefold. First, they can create yet missing data, thus establishing a means to plan ahead of knowledge. Second, they can predict which transports may be not on time. Third, they can surmise time deviations.

A. Neural Nets to Complete the Feed

Data becomes worse - or even non-existent - the farther the look into the future. Thus, missing but plausible data has to be integrated into the feed. The corresponding DNN gets trained with historical data of planned arrival times of trucks. From this training data, the net creates fictional yet plausible data entries to complete the fragmentary known data. The thus enriched data constitutes one possible scenario to be analysed by simulation. As it is the first fully scheduled data set, it can be regarded as the base scenario.

B. Neural Nets for Classification

The associated DNN can learn from historical data which deliveries and dispatches were on time. Thus, it can predict the probability of a trip to be delayed or advanced. This is due to the neural nets capability to learn from underlying correlations. Such correlations may be interpreted as questions like:

- Are there shipping agents that often are late?
- Are there producers that always are early?
- Are midweek deliveries more reliable than ones on Mondays?

After establishing the probabilities of unpunctuality, alternative scenarios can be established. These scenarios incorporate differing yet still plausible delay and advance times. The corresponding schedules can then be compared to the base scenario.

C. Neural Nets for Regression

Knowing which delivery may be late is one side of the coin. The other is the actual time. The fitting DNN can make guesses about these times. Again, the capabilities of neural nets to represent correlations prove useful: Several different effects may overlap that may add up to massive delays. An example for this may be an unreliable hauler in the midst of winter on an early Monday morning. These time delta represent the last puzzle piece in creating alternative scenarios.

D. Implementation Insights

As a tool only provided via internet, the architecture of the *P-S.C-Cloud* consists of a JavaScript client and a PHP Server. The prototypical implementation of a ML component to conduct the previously described tasks is done with Python instead. The reason for this is the existence of a large number of ML libraries that can simply be combined. Especially the following packages are used [12]:

- **NumPy** provides data types and functions for easier handling of complex structures, such as vectors and matrices.
- **pandas** is designed for more complex structures and their easy handling. One strength is its extensive functionality for table structures.
- **Matplotlib** is used for visual analyses and plotting.
- **scikit-learn** contains many ML algorithms that can be easily used in your own program.
- **Keras** can build artificial neural networks.
- **TensorFlow** extends Keras with additional well performing functionalities and can handle large and complex data structures.

With the goal to train a DNN as an example, typical delivery information including possible delays, have been guessed in a students' project. While some of the information have been randomised, others include pattern like specific shipping agents always being late. The implementation for this is explained next in Section VI in more detail.

The numerical representation of the input data is partially generated in the JavaScript part of the implementation, others make use of the predefined ML algorithms in Python.

The derived information concerning completed order lists, classification and regression are currently produced in the Python environment. They can be stored in CSV files which can then be integrated into the *P-S.C-Cloud* via file upload. The corresponding implementations are explained in Section VII and Section VIII.

VI. Synthetic Data

The data generation is the first step of the implementation. Like the other two steps, the implementation is explained in great detail ensuring that anyone can rebuild the solution from scratch.

A. Description of Data

The following categories have been build:

- **total:** the total number of pallets in a delivery which may influence the transport time.
- **producer:** manufacturer of the order delivery which might affect the departure time of a transport after loading and hence the punctuality of the arrival.
- **carrier:** carrier for the delivery of the order, because they may differ in terms of quality standards and delivery accuracy.
- **weekday:** the traffic intensity may vary at the different days of the week.
- **season:** seasonal differences of the weather conditions may influence the traffic.
- **distance:** the distance of a delivery from the manufacturer to the destination.
- **arrival:** the respective delivery time.
- **delay:** delay times in four classes. The aim is to predict delays depending on the aforementioned parameters.

The next step is to generate this data as a CSV file using Python code.

B. Import of Required Modules

The **pandas** and **NumPy** modules are to be imported at the beginning of the code.

```
# Import of required modules
import pandas as pd
import numpy as np
```

In this context, **NumPy** provides the tools for generating random values to simulate various input features of the data set.

The **pandas** library is utilized to organize the generated data into a structured DateFrame format which resembles a table. This format is suitable for further processing, exporting to a CSV file, and use in machine learning workflows. **pandas** allows for easy labeling of columns and transformation of categorical and numerical data into a unified structure.

C. Consistent Data Generation

The size of the data set is determined next. Also, the random seed is set to 42 to ensure that the generated data remains consistent each time the program is invoked.

```
# Determination of the size
number_of_records = 1000
np.random.seed(42)
```

Afterwards the categories and possible values are defined.

```
_number_of_pallets = np.random.randint(1, 24, size =
      number_of_records)
_producer_classes = ['prod1','prod2','prod3']
_carrier_classes = ['car1','car2','car2']
_weekdays = np.random.randint(1, 7, size =
      number_of_records)
_seasonal_classes = [1, 2, 3, 4]
_seasons = []
_transport_distance_classes = [250,500,750]
_transport_distances = []
_arrival_time_classes = ['morning','midday','evening']
_arrival_times = []
_delay_time_classes = [0,15,30,45]
_delay_times = []
```

In the subsequent phase, the carriers are allocated according to their associated probabilities. Each carrier is generated with an identical probability, ensuring an even distribution.

```
_carriers = np.random.choice(_carrier_classes,
      number_of_records, p=[0.3333333333333333,
      0.3333333333333333, 0.3333333333333333])
```

D. Delay Times, Transport Distances, and Arrival Times

After the carriers are assigned, delay times are generated. The objective is to create shorter delays for `car1`, longer delays for `car2` and moderate delays for `car3`. By creating varying delay times for each carrier, patterns can be identified in the data, which are essential for the AI's predictions.

```
def generate_delay_time(_carriers):
  if _carriers == 'car1':
    return np.random.choice(_delay_time_classes, p=[0.6,
      0.2, 0.1, 0.1])
  elif _carriers == 'car2':
    return np.random.choice(_delay_time_classes, p=[0.1,
      0.1, 0.4, 0.4])
  elif _carriers == 'car3':
    return np.random.choice(_delay_time_classes, p=[0.1,
      0.4, 0.4, 0.1])

for i in _carriers:
  _delay_times.append(generate_delay_time(i))
```

Next, transport distances are assigned to the different carriers. `car1` records have a transport distance of 250 km, `car2` 500 km, and `car3` 750 km.

```
def generate_transport_distance(_carriers):
  if _carriers == 'car1':
    return _transport_distance_classes[0] # 250 km
  elif _carriers == 'car2':
    return _transport_distance_classes[1] # 500 km
  elif _carriers == 'car3':
    return _transport_distance_classes[2] # 750 km

for i in _carriers:
  _transport_distances.append(generate_transport_distance(i
      ))
```

The three carriers also vary concerning their arrival times with `car1` deliveries usually arrive in the morning, `car2` midday, and `car3` in the evening.

```
def generate_arrival_time(_carriers):
  if _carriers == 'car1':
    return _arrival_time_classes[0] # midday
  elif _carriers == 'car2':
    return _arrival_time_classes[1] # morning
  elif _carriers == 'car3':
    return _arrival_time_classes[2] # evening

for i in _carriers:
  _arrival_times.append(generate_arrival_time(i))
```

E. Generating Seasons

Season dependent delay times represent delays caused by weather conditions.

```
def generate_season(_delay_times):
  if _delay_times == 45:
    return _seasonal_classes[3] # Winter
  elif _delay_times == 30:
    return _seasonal_classes[2] # Autmn
  elif _delay_times == 15:
    return _seasonal_classes[0] # Spring
  elif _delay_times == 0:
    return _seasonal_classes[1] # Summer

for i in _delay_times:
  _seasons.append(generate_season(i))
```

F. Creating the Data Structure

To complete the data generation, the data structure is created as a DataFrame and afterwards exported as a CSV.

```
data = pd.DataFrame({
  'total' : _number_of_pallets,
  'firm' : np.random.choice(_producer_classes,
     number_of_records),
  'carrier' : _carriers,
  'weekday' : _weekdays,
  'season' : _seasons,
  'distance' : _transport_distances,
  'arrival' : _arrival_times,
  'delay' : _delay_times
})

data.to_csv('delivery_data_auto.csv', index=False)
```

VII. Classification

The task is to use the previously generated data as training and test data in order to classify a possible delay in future deliveries.

A. Import of Required Modules

To implement the neural network and prepare the data set for training, several essential Python libraries and modules are utilized:

```
import tensorflow as tf
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
import numpy as np
```

Section V includes a short description of these libraries.

B. Import CSV Files

To use the previously generated data, the CSV file is imported as a DataFrame.

```
path1 = "delivery_data_auto.csv"
data = pd.read_csv(path1, delimiter=',')

# Output of the first five rows
print(data.head())
```

Figure 4 shows the first five rows of the data set.

Id	total	firm	carrier	weekday	season	distance	arrival	delay
0	7	prod1	car2	4	4	500	midday	45
1	20	prod1	car3	3	4	750	evening	45
2	15	prod3	car1	3	2	250	morning	0
3	11	prod2	car2	1	3	500	midday	30
4	8	prod2	car1	6	2	250	morning	0

Figure 4. Output of the first five rows.

Also, the data to which the AI makes a prediction is imported as a DataFrame as exemplarily shown in Figure 5. The record describes an order of 16 pallets of producer `prod3` carried out by the freight carrier `prod2` in winter, arriving midday on a Wednesday with a delivery-distance of 500 km.

```
Id total firm  carrier weekday season distance arrival
0     16 prod3    car2       3      4      500  midday
```

Figure 5. Data to predict.

C. Delay Classes

The next step is to define classes for the various possible delays. Figure 6 shows the result.

```
classes = data['delay']
print('Classes: ' + classes.sort_values(ascending=True).
      unique())
classes = len(classes.unique())
print('Number of output classes: ' + classes)
```

```
Classes: [ 0 15 30 45 ]
Number of output classes: 4
```

Figure 6. Output of classes and number of classes.

To predict whether a delay might occur, delays must be classified. For this, the column with the delays is separated from the input variables and stored separately in a normalized form.

```
col_name = 'delay'
le = LabelEncoder()
col = le.fit_transform(data[col_name])

data = data.drop([col_name], axis=1)
```

D. Handling of Non-Numerical Data

Categories with non-numerical values - like firm, carrier and arrival - need to be converted into numerical values using one-hot-encoding (ohe). This binary coding increases the number of input variables for the next step.

```
conv_ohe = ['firm', 'carrier','arrival']
data = pd.get_dummies(data, columns=conv_ohe, dtype=float)
```

E. Training and Test Data

The last preparatory step is to divide the data into training and test data. A ratio of 80:20 is usually applied, where 80% are used for training and 20% for testing the model.

```
train_data, test_data, train_col, test_col =
      train_test_split(data, col, test_size = 0.2,
      random_state = 42)
```

F. Structure of the Artificial Neural Network

Keras sequential model is used for multi-class classification. It consists of a single stack of layers sequentially connected to each other.

```
model = tf.keras.Sequential([
  tf.keras.layers.Input(shape=(data.shape[1],)),
  tf.keras.layers.Dense(32, activation=tf.nn.sigmoid),
  tf.keras.layers.Dense(64, activation=tf.nn.sigmoid),
  tf.keras.layers.Dense(classes, activation=tf.nn.softmax)
])
```

- The input layer acts as the entry point for the data into the neural network.
- Two fully connected hidden layers build the inner neuronal structure. The first layer contains 32 neurons, and the second has 64. Both use the sigmoid activation function to map the input values between 0 and 1.
- The output layer has as many neurons as there are classes in the classification task and the softmax activation function selects the class with the highest probability as the predicted output.

G. Configure the Learning Process

The `model.compile()` function configures the learning process. This step involves specifying optimizer, loss function and evaluation metrics. The following setup was used in this study:

```
model.compile(optimizer='adam', loss='
      sparse_categorical_crossentropy', metrics=['accuracy
      '])
```

The network is trained using the *Adaptive Moment Estimation* (Adam) optimizer, a widely used gradient-based optimization algorithm. Adam combines the benefits of momentum and adaptive learning rates by maintaining running estimates of both, the first (mean) and second (uncentered variance) moments of the gradients. This makes Adam well-suited for training deep neural networks on noisy or sparse data sets and helps ensure stable convergence.

Given that the delay classes are encoded as integers, the `sparse_categorical_crossentropy` loss function computes the cross-entropy loss between the integer-encoded true labels and the probability distributions predicted by the network's softmax output layer.

The model's performance during training and evaluation is measured using it's accuracy. The metric calculates the proportion of correctly predicted delay classes relative to the total number of predictions. Accuracy provides an intuitive measure of the model's ability to correctly classify future delay categories based on the input features.

H. Execution of the Training Process

The training process of the neural network is conducted using the `model.fit()` function, which initiates the learning phase. The following configuration is applied:

```
model.fit(train_data, train_col, epochs=60)
```

The model is trained with the input feature matrix (`train_data`) and the target vector `train_col` containing the integer-encoded delay categories.

The training runs for 80 epochs to optimize the model's internal weights, reduce classification error, and avoid underfitting. Over successive epochs, the model is expected to converge to a solution that generalizes well to unseen data, and accurately classifying future delay categories based on new feature inputs.

Figure 7 provides a summary of the model's performance during the final five training epochs.

```
Epoch 75/80
25/25 [==============================] - 0s 1ms/step - loss
      : 0.2887 - accuracy: 0.9488
Epoch 76/80
25/25 [==============================] - 0s 1ms/step - loss
      : 0.2816 - accuracy: 0.9563
Epoch 77/80
25/25 [==============================] - 0s 1ms/step - loss
      : 0.2733 - accuracy: 0.9638
Epoch 78/80
25/25 [==============================] - 0s 1ms/step - loss
      : 0.2634 - accuracy: 0.9762
Epoch 79/80
25/25 [==============================] - 0s 983us/step -
     loss: 0.2551 - accuracy: 0.9762
Epoch 80/80
25/25 [==============================] - 0s 1ms/step - loss
      : 0.2507 - accuracy: 0.9850
```

Figure 7. Output of the training process.

The final training epochs show a consistent decrease in categorical crossentropy loss and a steady increase in classification accuracy. By the end of training, a training accuracy of 98.50% was acieved, indicating effective learning and convergence.

I. Testing

To complete the training, the model was evaluated on a test data set to assess its generalization performance. This evaluation was conducted using the following configuration. Figure 8 depicts the result of the test process.

```
75 test_loss, test_acc = model.evaluate(test_data, test_col)
76 print('Test accuracy:', test_acc)
```

```
7/7 [==============================] - 0s 975us/step - loss
      : 0.2226 - accuracy: 0.9800
Test accuracy: 0.9800000190734863
```

Figure 8. Output of the test process.

`model.evaluate()` computes the final loss and classification accuracy on the test data. These metrics provide insight into how well the model performs on previously unseen samples and help determine whether the network has successfully generalized beyond the training set.

The test accuracy reflects the proportion of correctly predicted delay categories in the test data set. This step is crucial to validate that the high training accuracy observed during the final epochs is not a result of overfitting.

The model achieved a test accuracy of 98.00% with a corresponding test loss of 0.2226 which indicates that the model has successfully generalized beyond the training set.

J. Make a Prediction

To evaluate the usability of the trained model, a separate test data set (`delivery_test_ohe.csv`) was used.

```
79 path2 = "../Data/delivery_test4.csv"
80 data_to_predict = pd.read_csv(path2, delimiter=';')
81
82 print('Data to predict: ')
83 print(data_to_predict)
```

To verify the correct alignment of feature values with the expected schema, the test data sample used for predictions was printed to the console as shown in Figure 9.

```
Data to predict:
   total  weekday  season  transport-distance  firm_prod1
      firm_prod2  \
0     16        3       4                 500        0.0
             0.0

   firm_prod3  carrier_car1  carrier_car2  carrier_car3  \
0         1.0           0.0           1.0           0.0

   arrival-time_evening  arrival-time_midday  arrival-
      time_morning
0                   0.0                  1.0
         0.0
1/1 [==============================] - 0s 58ms/step
```

Figure 9. Output of the customised CSV file for the prediction

The test data set was manually created and aligned with the one-hot-encoding-training data. Now, it could be used to perform a prediction:

```
86 pred = model.predict(data_to_predict)
87
88 print('prediction in %: ', pred)
```

The neural network returns a probability distribution across all output classes, indicating the model's confidence in each potential delay-time category. Figure 10 shows the prediction for the given input:

```
prediction in %:  [[4.4574207e-03 3.2497539e-06 1.8211146e
      -01 8.1342787e-01]]
```

Figure 10. Output of the prediction in per cent.

The array contains four floating-point values that sum approximately to 1.0. Each value represents the predicted likelihood of the input belonging to the respective class, i.e., 0.45% for class 0, 0.0003% for class 1, 18.21% for class 2, and

81.34% for class 3 indicating that the order will be 45min late.

The highest probability value of the prediction array was determined using the **NumPy** function `argmax()` and the result is shown in Figure 11.

```
# Determination of the index of the largest number
num = np.argmax(pred)
print('Index value: ', num)
print('_' * 75)
```

```
Index value:  3
```

Figure 11. Output of the prediction as an index value.

To decode the index it must be mapped back to its original string using the inverse transformation of the `LabelEncoder`:

```
# Determination of the result as string
spec = le.inverse_transform([num])
print('Maximum expected delay time: ', spec, 'min')
print('_' * 75)
```

`le` refers to the previously fitted `LabelEncoder` instance, and `num` is the predicted class index obtained via `np.argmax(pred)`. `inverse_transform[num]` converts this numerical index back into its original categorical form. The result is shown in Figure 12.

```
Maximum expected delay time:  [45] min
```

Figure 12. Output of the prediction.

This completes the model inference pipeline, converting raw input data into a meaningful delay-time prediction.

The models decision is influenced by multiple factors in the input:

- Carrier `car2` was associated with a significantly higher probability of delay (30 or 45 minutes).
- The season value of 4 corresponds to winter with further potential delays.
- The arrival time of “midday“ which frequently co-occurred with `car2` deliveries and longer delays.
- A transport distance of 500 km which was also characteristic of `car2` deliveries in the synthetic data.

VIII. Regression

A regression model is implemented to predict continuous numerical values - specifically, the expected delivery delay in minutes. The foundation for this model is based on the previously developed classification pipeline. With minor modifications it can be adapted to perform regression tasks.

A. From Classification to Regression

Data preprocessing from the classification section remains unchanged. This includes label encoding, one-hot-encoding for categorical features, and splitting the data set into training and test sets. Only the configuration of the artificial neural network and the loss function require adjustments. While input and inner layers stay unchanged, the output layer is defined as a single neuron without an activation function in order to predict continuous values.

```
model = tf.keras.Sequential([
  tf.keras.layers.Input(shape=(data.shape[1],)),
  tf.keras.layers.Dense(32, activation=tf.nn.sigmoid),
  tf.keras.layers.Dense(64, activation=tf.nn.sigmoid),
  tf.keras.layers.Dense(1)
])
```

B. Configuration of the Learning Process

Also, the learning process uses the Adam optimizer for regression. Now, mean-absolute-error (mae) is chosen as loss function. It measures the absolute, average magnitude of the errors between predicted and actual values. mae is also used as the performance metric to monitor the accuracy:

```
model.compile(optimizer='adam',loss='mae',metrics=['mae'])
```

C. Training Process

The training process is almost initiated in the same way as for classification. Except the epoch number is raised to 80 for a better adjustment of internal weights for regression. The training performs as shown in Figure 13.

```
model.fit(train_data, train_col, epochs=80)
```

```
Epoch 75/80
25/25 ------------------ 0s 1ms/step - loss: 0.2938 - mae:
      0.2938
Epoch 76/80
25/25 ------------------ 0s 1ms/step - loss: 0.3143 - mae:
      0.3143
Epoch 77/80
25/25 ------------------ 0s 1ms/step - loss: 0.3102 - mae:
      0.3102
Epoch 78/80
25/25 ------------------ 0s 1ms/step - loss: 0.2562 - mae:
      0.2562
Epoch 79/80
25/25 ------------------ 0s 1ms/step - loss: 0.2742 - mae:
      0.2742
Epoch 80/80
25/25 ------------------ 0s 1ms/step - loss: 0.2950 - mae:
      0.2950
```

Figure 13. Output of the prediction as an index value.

These results indicate that the model achieved a relatively stable and low mae during the final training phase, suggesting effective convergence. The fluctuations in the mae values are minor and typical for this type of training.

D. Testing

The model's performance was evaluated on beforehand unseen test data. Figure 14 shows the result.

```
16 test_loss, test_mae = model.evaluate(test_data, test_col)
17 print('Test mae:', test_mae)
```

```
7/7 ------------------ 0s 4ms/step - loss: 0.2826 - mae:
      0.2826
Test mae: 0.2971033751964569
```

Figure 14. Output of the prediction as an index value.

For 7 batches, the output shows a loss of 0.2826. This indicates that the average absolute difference between predicted and actual delay times is approximately 0.2826 minutes per instance. Also the `test_mae` metric delivers a comparable value. The small deviation between the two values stems from internal rounding variances.

E. Make a Prediction

Once the hyperparameters for a regression have been adjusted and the model has been trained and tested, the AI is ready to make predictions.

```
20 path2 = "../Data/delivery_test4.csv"
21 data_to_predict = pd.read_csv(path2, delimiter=';')
22
23 print('Data to predict: ')
24 print(data_to_predict)
25
26 # Make a prediction
27 pred = model.predict(data_to_predict)
28
29 print('delay time in min: ', pred)
```

```
Data to predict:
   total  weekday  season  transport-distance  firm_prod1
      firm_prod2  \
0     16        3       4                 500         0.0
            0.0

   firm_prod3  forwarder_car1  forwarder_car2
      forwarder_car3  \
0         1.0             0.0             1.0
      0.0

   arrival-time_evening  arrival-time_midday  arrival-
      time_morning
0                   0.0                  1.0
         0.0
1/1 ---------------- 0s 46ms/step
_____________________________________________
delay time in min:  [[3.0888479]]
_____________________________________________
```

Figure 15. Output of the prediction in minutes.

IX. ML Extended Simulation

It is the aim to integrate the ML solution presented in the previous sections into the formerly introduced simulation environment as shown in Figure 16. After the trusted orders are entered for simulation, a new phase **refurbish** is executed which makes use of the ML model.

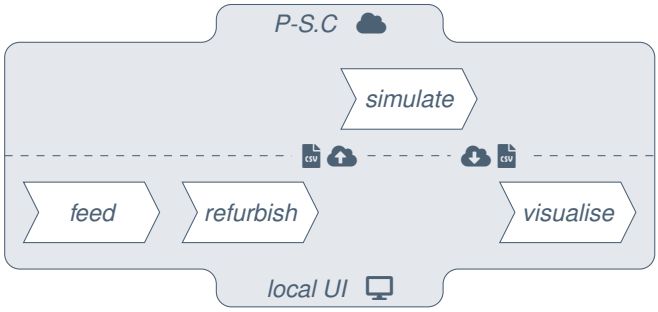

Figure 16. Simulation environment extended by ML.

With the goal to improve the ramp allocation, simulation is conducted in two phases. Figure 17 shows the improved planning approach.

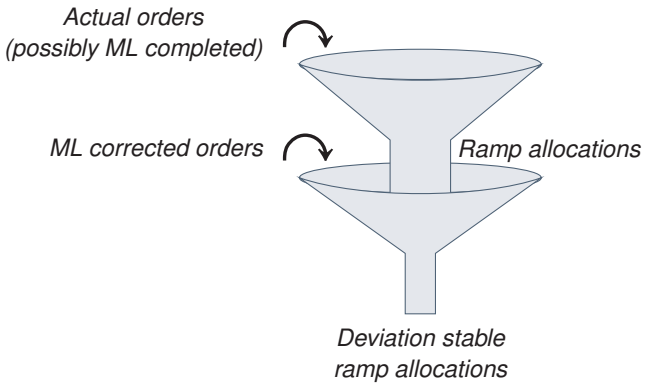

Figure 17. Two-Phase approach for ramp allocation.

A. Simulation of a Planned Schedule

In the first iteration, the orders are simulated for the case that all transports arrive as planned. If further orders are expected for the simulation period, the ML algorithm is able to generate plausible orders with respect to the known order history.

The simulation is conducted with the goal to find an optimal ramp allocation which reduces idle times for the shipping companies combined with a minimal labour utilisation. Different ramp allocations may lead to almost same results.

B. Simulation of a Planned Schedule plus (DNN) Delays

During the second phase, the orders are corrected with respect to the assumed deviations of transport times. The ramp allocation strategies found in the first phase are applied

to these adapted schedules and evaluated with respect to the optimisation criteria. These simulation results are used as a filter to find that allocation strategy which fits best to the original and the adapted orders.

C. Preliminary Work of Industry Partners

As mentioned before, the industry partner does not collect information needed for the machine learning approach. To prepare a data collection, the training data mentioned before is divided into three classes:

- **Available**: timestamp of the planned arrival; incoming vs. outgoing transports; amount of pallets
- **Not available but collectable**: the delay of transports; transportation distance in the sense of transports inside the industry park or nor; the shipping agent; the producer
- **Not available and not collectable**: in advance, the transportation distance in length

This classification reinforces the appraisal that the introduced approach can be conducted. It's application depends on the willingness of the industry partners.

X. Conclusion and Outlook

As initially stated, the goal of this research is to provide an environment to improve simulation in logistics. It uses machine learning based techniques to create schedules that are (mostly) indifferent to deviations from the original planning. Thus, a good schedule performs at least as good on the original data as the original schedule; but it performs as good as possible under differing scenarios.

A prototypical environment has been implemented, but the parts are not fully integrated yet. The reason for this is that the industry partner does not collect information of the deviation of planned and actual arriving time of the trucks. Also, other information that could help to train a neural net are not considered by the partner. The possibility to do so, has been explained above.

This represents the next important research step: check the approach with real-world data! Afterwards it makes sense to develop the integration further. The necessary implementation details have been uncovered in this article.

Even with this approach it is not possible to know which transports will be late in the future. But the developed scenario is more stable with respect to delays in delivery without degrading the initial plan. And for checking this plan, simulation is still needed. Purely statistical estimations are insufficient for processes in logistics.

References

[1] C. Simon, S. Haag, and N. G. Badurasvili, "Predictive AI To Feed Simulation", in *SIMUL 2024: The Sixteenth International Conference on Advances in System Simulation*, Venice (Italy), 2024, pp. 58–63.

[2] L. Zakfeld, C. Simon, M. Hladik, and S. Haag, "Real World Case Study To Teach Simulation", in *SIMUL 2023: The Fifteenth International Conference on Advances in System Simulation*, Valencia (Spain), 2023, pp. 19–24.

[3] Infraserv Logistics GmbH, *Overview hazardous substances warehouse*, https://www.infraserv-logistics.com/en/isl/news/news/ (last accessed 08.2024), 2023.

[4] C. Simon and S. Haag, "Pairing Finite Automata and Petri nets - Simulation of Processes in Logistics", in *ECMS 2024: 38th International ECMS Conference on Modelling and Simulation*, D. Grzonka, N. Rylko, and G. S. V. Mityushev, Eds., Krakow (Poland), 2024, pp. 474–480.

[5] C. Simon, S. Haag, and L. Zakfeld, "The Process-Simulation.Center", in *SIM-SC: Special Track at SIMUL 2022: The Fourteenth International Conference on Advances in System Simulation*, F. Herrmann, Ed., Lisbon (Portugal), 2022, pp. 74–77.

[6] T. Bergs *et al.*, "The Concept of Digital Twin and Digital Shadow in Manufacturing", *Procedia CIRP*, vol. 101, pp. 81–84, 2021, 9th CIRP Conference on High Performance Cutting, ISSN: 2212-8271.

[7] S. Russell and P. Norvig, *Artificial Intelligence - A Mordern Approach*, 4th ed. London: Pearson, 2021.

[8] P. Flach, *Machine Learning - The Art and Science of Algorithms that Make Sense of Data*, 9th ed. Cambridge: Cambridge University Press, 2012.

[9] J. Howard and S. Gugger, *Deep Learning for Coders with fastai & PyTorch*. Sebastopol: O'Reilly, 2020.

[10] A. Géron, *Hands-on Machine Learning with Scikit-Learn, Keras & TensorFlow*. Sebastopol: O'Reilly UK Ltd., 2019.

[11] C. Mattmann, Ed., *Machine Learning with TensorFlow*, 2nd ed. Shelter Island, NY: Manning, 2020.

[12] M. Karatas, *Development of AI applications (in German: Eigene KI-Anwendungen programmieren)*. Bonn: Rheinwerk Computing, 2024.