Energieforschungsprogramm

Publishable Report

Program control:
Klima- und Energiefonds

Program execution:
Österreichische Forschungsförderungsgesellschaft mbH (FFG)

Final Report
created
2024-02-29

Factories4Renewables

Project number: 881136

Ausschreibung	6. Ausschreibung Energieforschungsprogramm
Projektstart	01/04/2021
Projektende	29/02//2024
Gesamtprojektdauer (in Monaten)	34 Monate
ProjektnehmerIn (Institution)	Technische Universität Wien, Institut für Computertechnik
AnsprechpartnerIn	Thilo Sauter
Postadresse	Gusshausstraße 27-29/E384, 1040 Wien
Telefon	+43 1 58801 38430
Fax	+43 1 58801 38499
E-mail	thilo.sauter@tuwien.ac.at
Website	https://www.ict.tuwien.ac.at/

Factories4Renewables

Optimization of industrial production processes towards 100% renewable energy usage

AutorInnen:
Julian Binz, PowerSolution Energieberatungs GmbH
Aleksey Bratukhin, University for Continuing Education Krems
Alireza Estaji, University for Continuing Education Krems
Simon Howind, TU Wien
Stefan Kollmann, University for Continuing Education Krems
Roland Stenzl, University for Continuing Education Krems
Martin Weinzerl, AVL List GmbH

1 Table of Contents

Table of Figures

Table of Tables

Table of Equations

2 Introduction

Today, industrial production planning is driven by the goal of optimal use of production facilities, considering the time constraints for individual orders. Considerations for optimized energy usage play only a minor role, if at all, e.g. in avoiding peak loads.

However, the ambitious goal of changing Austria's electricity supply to 100% renewables (#mission 2030) changes the structure of energy production considerably. At the same time, the total demand for electricity is expected to increase considerably by 2030. Many areas that currently still operate on fossil energy are gradually changing to electricity. This ranges from heating systems to process heat generation to mobility. A higher electricity demand together with the inevitable volatility of renewable energy means that production planning must consider the availability of energy in order to achieve the primary goals of #mission 2030. With the expansion of renewable energy sources, the baseload share of electrical power plants decreases, and energy production becomes increasingly volatile. Under these circumstances, a fixed load profile is often no longer realistic; instead, the load should be adapted to the available energy.

In a liberalized energy market, the availability of energy is just a question of price that the customers are willing to pay. If there is a demand, there is going to be a supply. Although it is envisioned that in the future, electricity markets will drift away from a purely demand-supply driven paradigm, in the foreseeable future, it can be assumed that the demand will always be satisfied. Therefore, the conversation of energy optimization shifts from energy availability towards the energy costs.

The primary objective and innovation of the Factories4Renewables concept is to enable a transparent integration of the energy costs into the production planning process driven by the dynamic uncertain energy prices. Because the companies tend to be unwilling to change their production planning systems due to the high associated costs, the goal of the approach was to integrate systems under the umbrella of production planning in a transparent fashion. On the one hand, that allows to use any type of already existing applications as well as to integrate the energy awareness with minimal additional costs to the system consumer.

In a nutshell, the optimization algorithm developed for the approach puts restrictions for the state-of-the-art production planning system, by shifting the available energy from expensive to cheap time slots. In the production planning system, the energy is integrated via a concept of Virtual Energy Machines (VEM) that are used by the production scheduling tools in the same way as any other type of machinery and added to any production step that requires energy use. By controlling the availability of VEMs, the system allows integration of the energy cost optimization without the need for changing production planning. To facilitate such integration, the system requires information about the energy consumption for execution of the production steps that is facilitating via modelling of each production step. In addition, the system needs to

account for the risks that are associated with the uncertainty of both energy prices and energy consumption, since both are modelled values and will deviate from reality.

3 System Architecture

The fundamental idea is to restrict the scheduling process of the MES software by limiting the available power at each time. The energy-aware scheduling process comprises four tasks. The first task involves predicting the likelihood of price predictions. The second task requires creating energy profiles for all processes. Scheduling is performed by the MES system. The main function of the Confidence Estimator is to provide energy constraints to the production scheduling in a transparent fashion by restricting the operational space for scheduling methods. These tasks are processed in four modules:

- Entropy Regulator (ER) - Calculates the probability of energy prices following a specific price profile.
- Confidence Estimator (CE) - Interacts with the scheduling software and impacts the scheduling calculations.
- Energy Module (EM) - Determines the energy profiles for each work step.
- Manufacturing Execution System (MES) - Creates the schedule

The interaction between these four modules is illustrated in Figure 1.

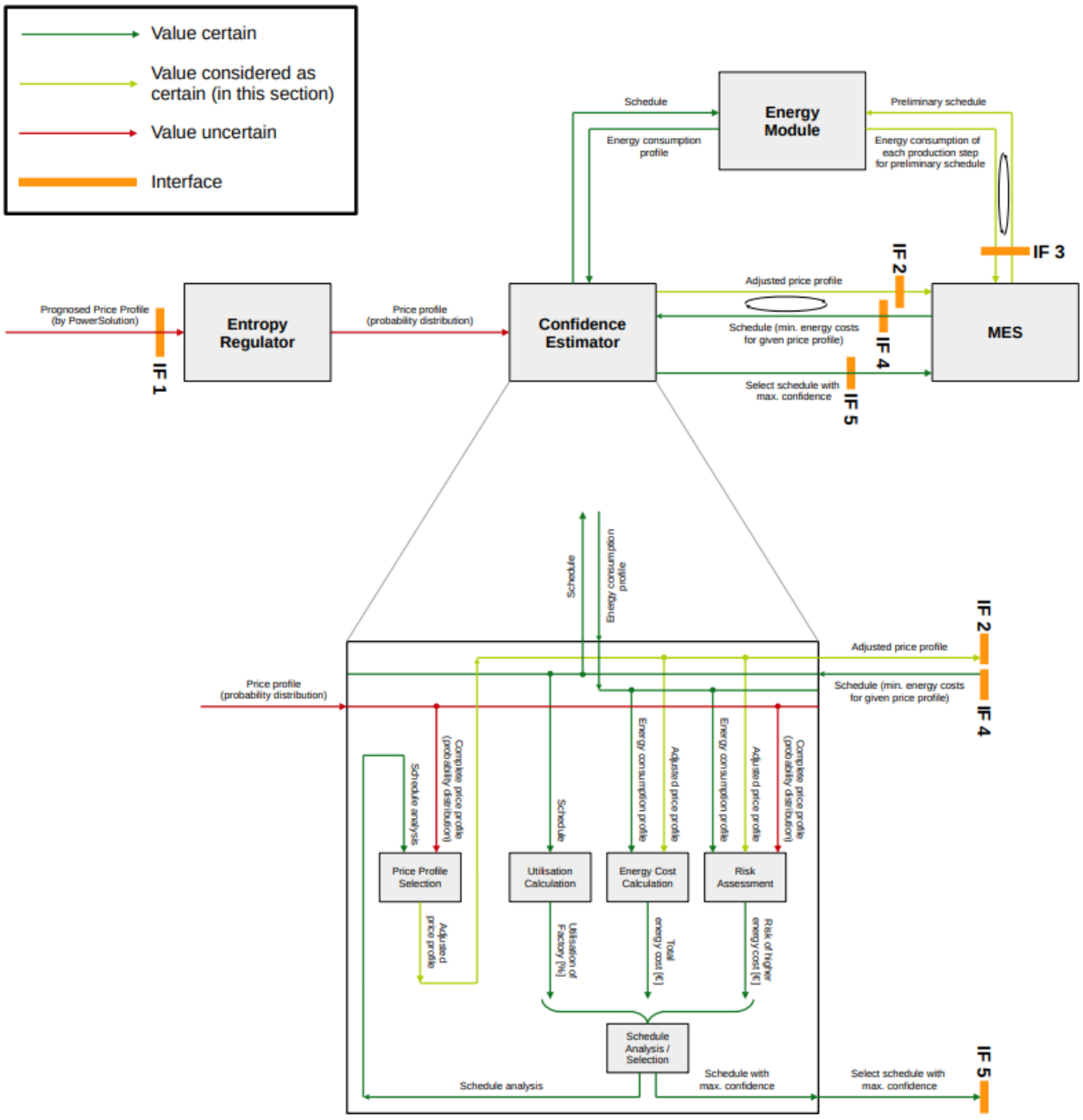


Figure 1 System architecture

3.1 Entropy Regulator1

Uncertainty in energy prices mostly relates to the volatility of renewable sources. Research on handling this volatility is indirectly addressing issues with energy price prediction uncertainty because it intends to mitigate the volatility and thereby the price uncertainty. Meanwhile, there is a variety of research focusing

[1] This section is an excerpt from a publication Bratukhin, Aleksey, et al. "Probability-based, Risk-adjusted Energy Consumption Optimisation in Industrial Applications." 2022 IEEE 27th International Conference on Emerging Technologies and Factory Automation (ETFA). IEEE, 2022.

on generation capacity prediction and the associated uncertainty of the energy availability. Examples of common methodology in uncertainty modeling are possibilistic modelling, interval-based analysis, probabilistic modelling that includes analytical modeling, e.g., based on approximated distributions or linearization, and numerical methods such as Monte Carlo Simulation, robust optimization, information gap decision theory, and hybrid probabilistic and possibilistic, e.g., fuzzy and scenario-based approach as well as fuzzy and Monte Carlo Simulation. In essence, uncertainty modelling generates large numbers of different scenarios, where each scenario represents a possible realization of the underlying uncertainties. The goal is to find the closest approximation of uncertainties true distribution characteristics by inferring a probability distribution on a parameter of interest based on a given probability density function (PDF). The probabilistic approaches are the traditional and most used approaches. However, a synthesis of traditional and modern methods usually provides best results for uncertainty modeling in applications. Analytical methods aim to quantify the uncertainty by applying known statistical qualities of the prior distributions to estimate the confidence intervals and likelihoods of certain outcomes. The most common numerical models are based on the Monte Carlo Simulation (MCS) method, which is based on the iterative sampling of data from the prior distribution to determine the population distributions. MCS methods are generally classified into three groups: sequential, pseudo-sequential and non-sequential, which differ in their approach to the chronological order of data processing and the related effect on the computational resources required. MCS is often used in a hybrid approach with algorithms that reduce the possible scenario generation space to further reduce the resources required to perform the MCS.

The Entropy Regulator analyzes the distribution characteristics of passed energy prices' prediction deviations and calculates risk adjusted prices. Prior to the operation phase it needs to be trained using historical data to determine the distributions to be expected for different time slots. In the operation phase it calculates a range of price profiles for different deviations covering probabilities which are subsequently used by the confidence estimator for the optimization. Prior to optimization, the Entropy Regulator must estimate distributions for each time slot up to the planning horizon. Later, when it receives a new price prediction it calculates the ranges for different probabilities that can be converted into risks as shown in Figure 1.

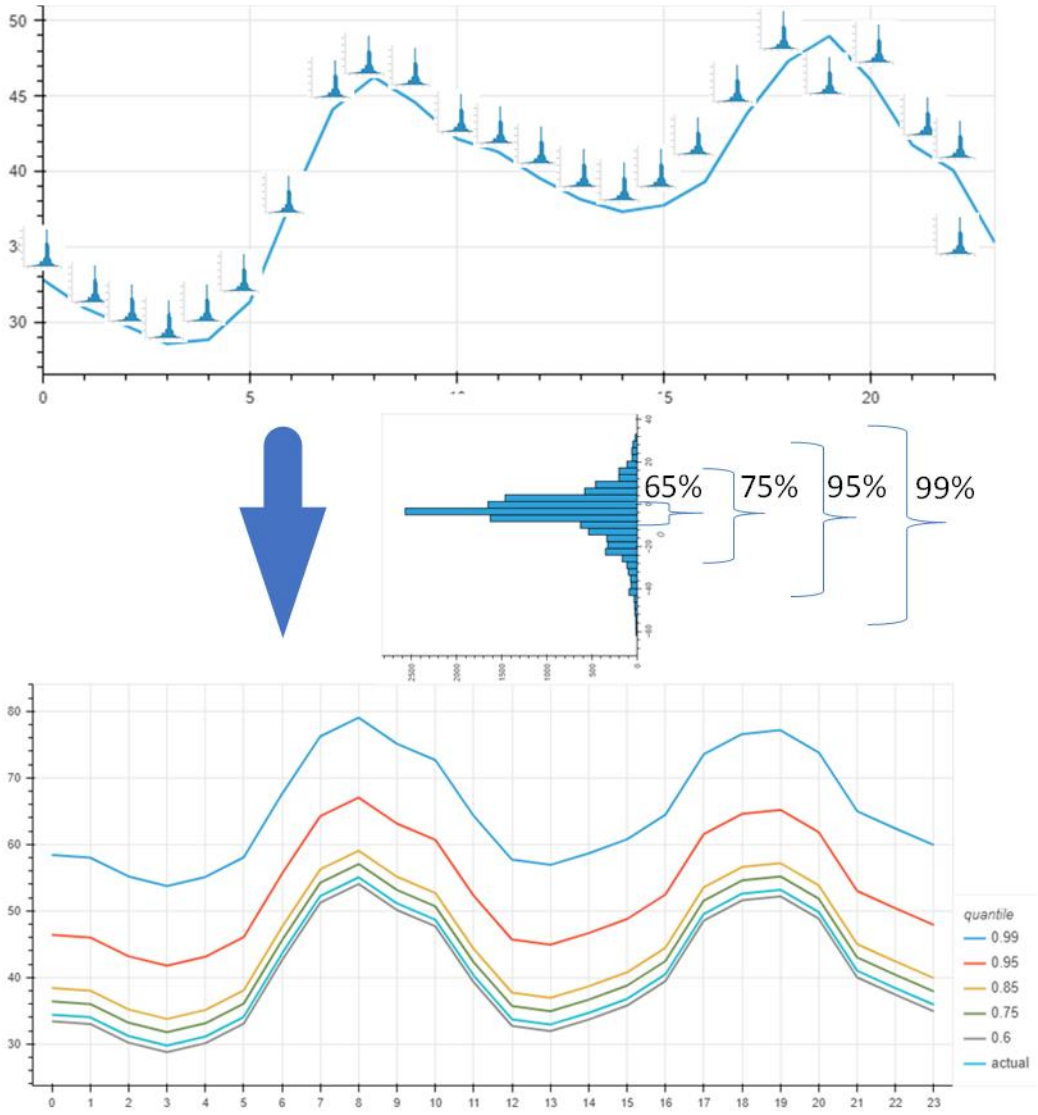


Figure 1: Entropy Regulator operation

The accuracy of the Entropy Regulator estimations highly depends on how accurate the assumed distributions, i.e., the modeled stochastic process, represent the reality of energy price prediction deviations.

Due to intellectual property (IP) protection rights, the actually used price prediction model generally will not be available for the Entropy Regulator as it is commonly considered a classified company secret, i.e., a business foundation. Neither are historical prediction records easy to get because the prediction service providers fear reverse engineering. The only performance metric revealed are the distribution characteristics of the deviation between the actual prices and the predicted prices.

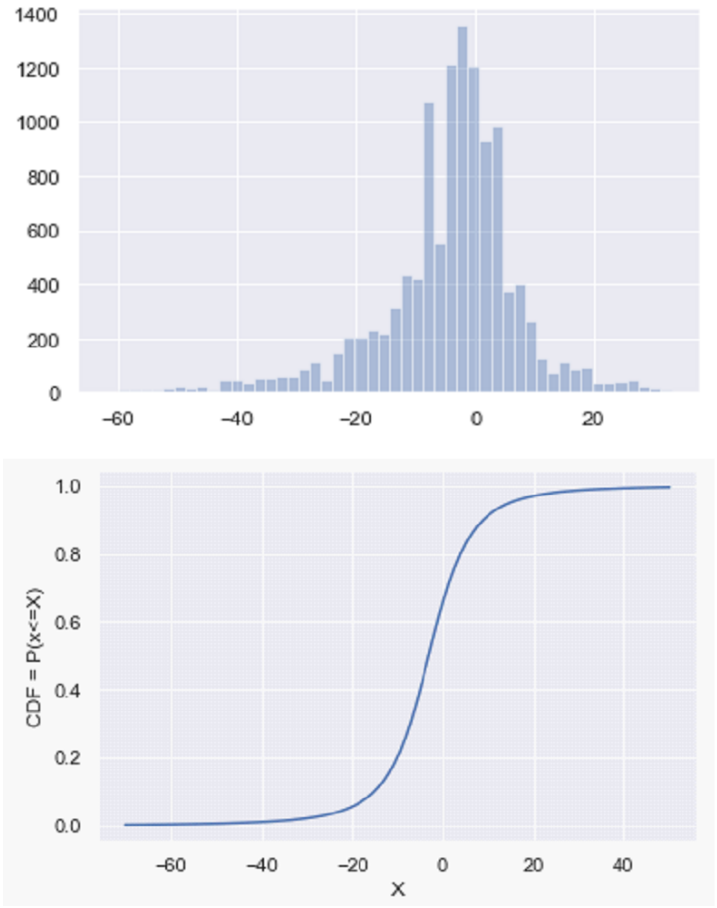


Figure 2: Price predictions deviation distribution

The original data records, on which the presented analysis is performed, are the electricity prices and the corresponding predictions for the year 2019, with a granularity of one hour. The analysis uses the actual deviations of the predictions from the actual prices only. Figure 2 shows the histogram, i.e., the sampled probability distribution function (PDF), and the cumulative distribution function (CDF) of the energy price prediction deviations. To perform statistical analysis and inference, an according "dataset" is generated based thereon.

The distributions generated by the Entropy Regulator can be used in two ways to estimate error/risk probabilities: (a) calculate them numerically using the sample dataset, and (b) try to find a fitting distribution function and use its known statistical properties (characteristics) to assess the error/risk probabilities of future production time slots.

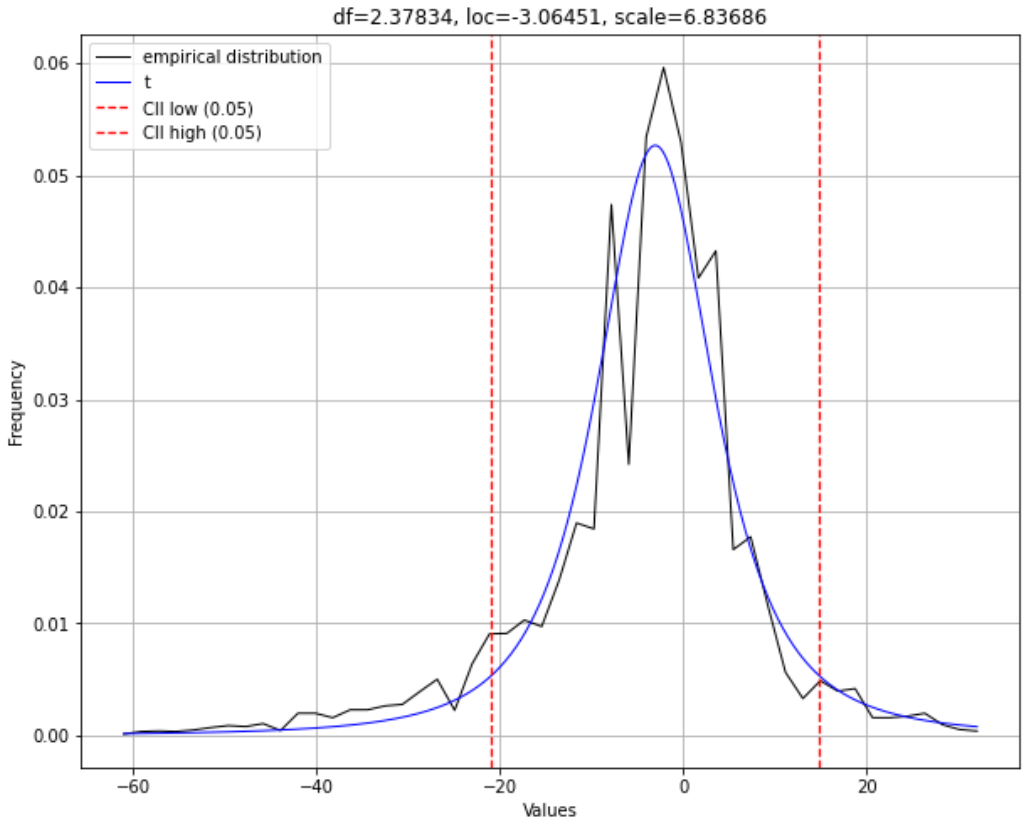


Figure 3: Fitting price predictions deviation distribution

For the latter, fitting a distribution function to the generated dataset uses the residual sum of squares method, which calculates the sum of the squares of residuals (deviations between actual and expected values) as shown in equation:

Equation 1: fitting a distribution function to the generated dataset

$$RSS = \sum_{i=i}^{n} \left(y_i - f(x_i)\right)^2$$

where y_i is the expected value, $f(x_i)$ is an observed value, and n is the size of the dataset.

The generated dataset fits the Student's-T distribution with the following parameters: mean = -3.06, standard deviation = 6.82, and degree of freedom = ,2.37. This fitting is shown in Figure 3 for generated data shortly after midnight, i.e., for prediction deviations expected in the first time slot (hour) after midnight. Other daytimes show little difference, probably due to the simplified test set-up assuming the prediction performs equally for all times of the day across all weeks and months of the year, i.e., by using the whole dataset available, which covers exactly one year, to train the Entropy Regulator. For this set-up, the mean error values vary from -3.02 to -3.08, the standard deviation from 6.78 to 6.86, and the degrees of freedom from 2.35 to 2.39.

The fitting of the dataset was performed on nearly 12000 samples of data for each hour of the day. However, with potential fragmentation of data into sub-classes to more accurately represent the reality of the energy prices and corresponding deviations, it is assumed that available datasets will be much smaller. To estimate the minimal sample size that provides acceptable accuracy, sample fitting on different sizes were performed and summarized in the table shown in Table 1.

Table 1: Comparison based on the sample

Student-T	10	12	26	25	32	33	40	40	45	44	46
Dweibull	15	24	18	21	18	17	10	10	5	6	4
Beta	15		1	1							
Lognorm		3	2	2							
Genextreme		2	1								
Gamma		2									
Pareto		2									
Uniform		1									
Loggamma			2	1							

The test was performed on 100 random samplings with different sample sizes, and at sample size 80, only T and Weibull distributions were detected (both perform in a similar way). Also, with increasing sample size the deviations of key parameters are reduced. Therefore, it can be estimated that the minimal sample size 80 is sufficient to correctly estimate the population distribution on the limited number of observations for price deviations.

The further use is based on the calculation of the predictions with associated error/risk probabilities. In case that the values have a certain probability for the entire planning period, the CDF of a detected distribution can be used to calculate the values of a certain probability. CDFs directly show the probability that a value is below a certain threshold value. Another option is to use the quintiles' function, based on the PDF or some histogram. Since the energy price deviations in the model fit the Student's,T distribution, several statistical mechanisms are available to estimate the risks based on the confidence intervals. If we assume that all prediction deviations have the same 'behavior' characteristics, there exist ample historical measurements to use this distribution for the probability estimations. However, in reality the prediction errors may vary depending on a variety of parameters: the hour of the day, days of the week, season, holidays, energy markets, electricity mix in the basket, how many days ahead the prediction was made, and so on. In that case, knowing the distributions, specifically, knowing that it is a t-distribution, allows the use of a limited number of samples to estimate the probabilities and calculate the risk associated with the prediction error. Using the known distribution, the risk, which is the cost of error in predictions that is proportional to the energy consumed multiplied by the deviation can be calculated.

3.1.1 Risk calculation

Risk is defined as the potential cost of error by the energy price prediction model. Since it is only relevant that the price is below a certain value, the risk for time slot i can be calculated as shown in equation

Equation 2: Risk as potential cost of error

$$Risk_i = \left(1 - \int_{i=0}^{Dev} p(x_i)\,dx\right) \cdot Con \cdot Dev$$

where $p(x_i)$ is the probability density function of the distribution, Con is the energy consumed in the time slot and Dev is the predicted price deviation, which yields the desired probability. It is defined as a quantile function, the inverse of the CDF, i.e., given by equation:

Equation 3: Quantile function for predicted price deviation

$$Dev = F^{-1}(p_{set})$$

where p_{set} is the desired probability and F is the CDF of the distribution. The total risk is the sum of risks of each time slot within the planning horizon, as shown by equation:

Equation 4: Risk summation

$$Risk_{total} = \sum_{i=0}^{n} Risk_i$$

where n is in this case the prediction time span in hours.
The Entropy Regulator then passes the risk adjusted price profiles to the confidence estimator to perform energy cost optimisations that account for the expectable error (i.e., its most likely distribution) in the modeled energy price predictions.

3.2 Energy Module

The purpose of the energy module is to derive a load profile from the production schedule, i.e., the full list of individual processes and their parameters. This way, each process is assigned a load profile and the schedule can be optimized accordingly. Figure 4 illustrates this process schematically.

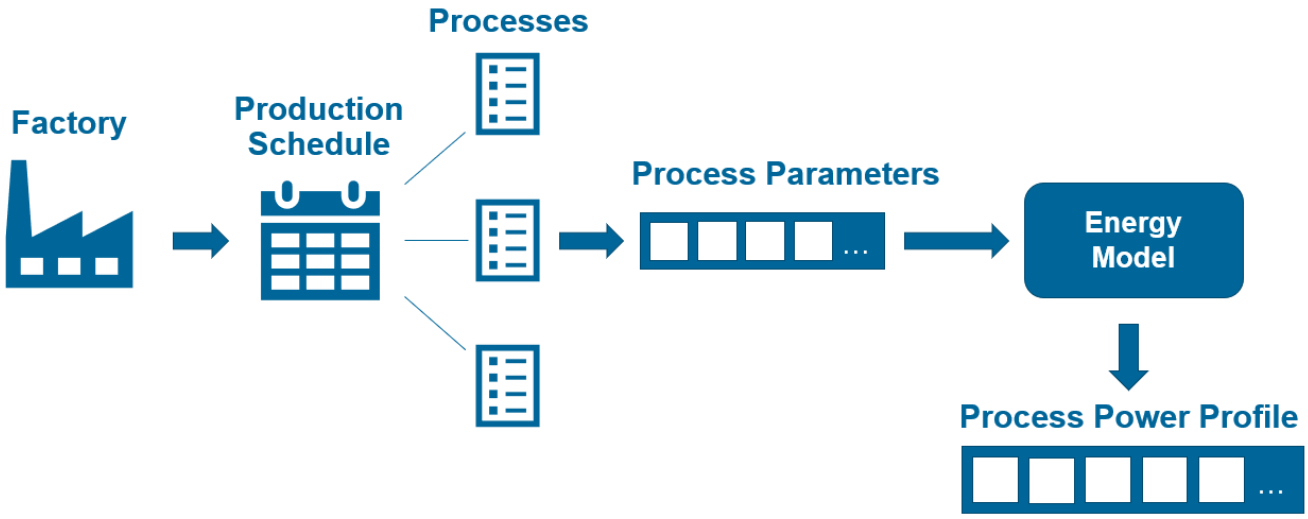


Figure 4: Schematic Function of the Energy Model

Numerous approaches exist to model the energy consumption of production processes with different focuses and strengths. Therefore, before selecting one approach or several approaches to compare against each other, it is necessary to clearly define the requirements and desired abilities for the Energy Model architecture with regard to forecasting the load profile of manufacturing processes and schedules. The first requirement for the model architecture is that it be easily adaptable to various industrial use cases. This means that by simply replacing the historical data and performing automated procedures, the model should work in a different use case. The second requirement is that the model can interpolate between parameter values from the historical data. This ensures that the model also performs well in dynamic and flexible production environments, e.g., with customizable products or parameterizable machines. Lastly, the third requirement is that the model encompasses all process-dependent energy consumption, i.e., the energy consumption of all operations that are performed on the manufacturing machines. This includes idle consumption since it is not just some constant base load but depends on the timing of scheduled processes and on the duration of idle periods between processes, particularly for machines requiring

standby mode during interventions. Scheduling processes consecutively followed by a longer idle period allows for complete machine shutdown during the break and, consequently, a lower idle energy consumption compared to multiple short idle periods scattered between the processes. This combination of requirements cannot be fulfilled with existing energy modeling approaches from the literature. Finite State Machine (FSM) models, while being adaptable to various industrial use cases, cannot interpolate between parameter values from the historical data. Specific physical models can also handle unprecedented settings but require a lot of effort to develop, are tailored to a specific application, and are, therefore, hardly adaptable to various use cases. Data-driven approaches currently only exist for specific use cases, to parameterize model-driven approaches, or for factory-wide energy consumption. Therefore, by combining existing approaches from other application domains, a novel Energy Model has to be developed. The major decision that has to be made is the one between a model-driven or a data-driven approach. The strengths of model-driven approaches include high accuracy and interpretability. Especially interpretability is an important advantage over data-driven approaches since it enhances trust and reliability in the model. Moreover, examining the model's inner workings enables insights for discovering hidden drivers of energy spikes beyond the obvious drivers and for the development of scheduling-specific optimization strategies. By understanding how process timing and process sequence impact energy consumption, production plans can be adapted to minimize, e.g., idle periods and optimize machine states. Model-driven approaches also fulfill the requirements of being able to interpolate between known data points as described above. A model based on principles of physics, thermodynamics, or engineering can make predictions for scenarios not explicitly present in the training data by using the underlying mathematical or physical principles that describe the system. Also, idle times can be modeled just like production processes. However, with regard to the requirement of adaptability described above, data-driven approaches have the advantage. Their strengths include flexibility and adaptability; new data and information can be easily integrated. They require much less domain-specific knowledge and understanding of the underlying principles. At the same time, they are able to interpolate between known data points as long as they are not too far apart, which is also the main reason for not just using historical load profiles without further treatment as a forecast for the load profile of processes. Additionally, idle times, as long as they are present in the historical data, can be treated and modeled like production processes. This makes data-driven models strong candidates for the given use case, particularly in situations where domain-specific knowledge may be limited or where the system's behavior is complex and not fully understood.
Six model architectures are tested: a Neural Network, the Recurrent Neural Network architectures Long-Short-Term Memory (LSTM) and Gated Recurrent Unit (GRU), Encoder-Decoder Networks with LSTMs and GRUs, and an Ensemble Learning model. The ensemble model is a parameter-based ensemble model of several instances of the RNN LSTM architecture. The model is chosen based on the recipe ID, so each model specializes in the load profile forecasting of one process type only.

For the implementation of the model architectures, the Python library Keras was used. Keras is a high-level neural network Application Programming Interface (API) written in Python that provides an interface for building and training deep learning models. Keras is built on top of TensorFlow, an open-source numerical computation library, and uses TensorFlow's efficient backend for computations while providing a simpler and more intuitive API for model building. In the following subsections, the Keras implementations of each model architecture are described and explained. The different model architecture implementations are all subclasses of the class "EnergyModel". They differ only by their constructor (which sets the architecture name) and the function "build_model", which defines the Keras model.

3.2.1 Neural Network

The NN architecture consists of a sequential model composed of several dense layers. Each dense layer employs the rectified linear unit (ReLU) activation function, which introduces non-linearity into the network, allowing the model to learn complex patterns from the data. The model architecture is as follows:

```
model = Sequential()
model.add(Dense(256, kernel_initializer='normal', input_shape=input_shape, activation='relu'))
model.add(Dense(256, activation='relu')) # Layer 2
model.add(Dense(256, activation='relu')) # Layer 3
model.add(Dense(256, activation='relu')) # Layer 4
model.add(Dense(output_shape[0], kernel_initializer='normal', activation='linear'))
model.compile(loss='mse', optimizer='adam')
```

The input layer (implicitly defined) receives the raw data, represented by a two-dimensional array with the shape input_shape, where input_shape specifies the number of samples and the number of features, respectively. The NN comprises four dense layers, each with 256 neurons. Dense layers facilitate connections between all neurons in one layer and all neurons in the subsequent layer. The ReLU activation function is applied to the output of each layer. The final dense layer has output_shape[0] neurons to achieve the desired dimensionality of the output. The linear activation function is applied since it is a regression problem and a numerical output is required. The linear activation function leaves that output unchanged.

3.2.1.1 Recurrent Neural Network

The RNN architecture is implemented as a 2-layer model with dropout layers. Both LSTM and GRU versions were implemented. In this section, the LSTM version is described, but the LSTM layer and the GRU layer are interchangeable. The model architecture is as follows:

```
model = Sequential()
model.add(LSTM(64, input_shape=input_shape, return_sequences=True))
model.add(Dropout(0.2))
model.add(LSTM(64, return_sequences=False))
model.add(Dropout(0.2))
model.add(Dense(output_shape[0]))
model.compile(loss='mse', optimizer='adam')
```

The first LSTM layer has 64 units (memory cells) and utilizes the return_sequences=True parameter to maintain the original sequence length. This allows the input sequence to be passed through subsequent LSTM layers without being truncated. Adding dropout layers is a regularization technique that randomly ignores a certain percentage (20 % in this case) of neurons during training. This helps prevent overfitting and improves the generalization of the model. The second LSTM layer has the same number of units as the first layer and uses the return_sequences=False parameter to indicate that the output sequence length should be reduced to one value. Another dropout layer is applied to the output of the second LSTM layer, maintaining the regularization effect. The final layer is a dense layer with the same number of neurons as the desired output dimension and a linear activation function.

3.2.2 Sequence to Sequence Model

Just as for the RNN architecture, the Seq2Seq Model was also implemented as both a LSTM and GRU version. The LSTM implementation is based on the description in the Keras blog [Cho23] and is as follows:

```
# Models for training
encoder_inputs = Input(shape=(None, input_shape[1]))
encoder = LSTM(64, return_state=True)
encoder_outputs, state_h, state_c = encoder(encoder_inputs)
encoder_states = [state_h, state_c]
```

```
decoder_inputs = Input(shape=(None, output_shape[1]))
decoder_lstm = LSTM(64, return_sequences=True, return_state=True)
decoder_outputs, _, _ = decoder_lstm(decoder_inputs, initial_state=encoder_states)
decoder_dense = Dense(output_shape[1], kernel_initializer='normal', activation='linear')
decoder_outputs = decoder_dense(decoder_outputs)

model = Model(inputs=[encoder_inputs, decoder_inputs], outputs=decoder_outputs)
model.compile(loss='mse', optimizer='adam')
self.model = model

# Models for inference
encoder_model = Model(encoder_inputs, encoder_states)
decoder_state_input_h = Input(shape=(64,))
decoder_state_input_c = Input(shape=(64,))
decoder_states_inputs = [decoder_state_input_h, decoder_state_input_c]
decoder_outputs, state_h, state_c = decoder_lstm(decoder_inputs, initial_state=decoder_states_inputs)
decoder_states = [state_h, state_c]
decoder_outputs = decoder_dense(decoder_outputs)
decoder_model = Model([decoder_inputs] + decoder_states_inputs, [decoder_outputs] + decoder_states)
encoder_model.compile(loss='mse', optimizer='adam')
decoder_model.compile(loss='mse', optimizer='adam')
```

To accelerate the training, a method called teacher forcing is applied. This means that the output loopback to the input of the decoder is interrupted during training. Instead, the true output values are applied to the decoder input (shifted by one). To apply teacher forcing during the training, the inputs and connections between encoder and decoder LSTM differ between training and inference. The upper half of the Python code describes the connections for training, and the lower half describes the connections for inference. The encoder input is a vector with a length equal to the number of features. This means that all features are passed to the encoder in a single time step. Theoretically, it would also be possible that the features would be passed one after the other in several time steps. However, since there is no temporal relation between the process parameters (i.e., unlike a time series, the values have no “before” or “after” relation), this would be detrimental to the performance of the model. The encoder itself is an LSTM layer with 64 units. While the internal state is passed to the output because it is forwarded to the decoder, the actual encoder output is discarded. During training, the decoder takes as input the desired output sequence shifted by one (teacher forcing), the start token is the value −2. The decoder, just like the encoder, is an LSTM with 64 units which passes the internal state to the output. However, “return_sequences” is set to True to return the full sequence of outputs (instead of just the last one). This is because the output sequence of the decoder is actually used and not discarded like the output sequence of the encoder. Also, the initial internal state is set to the internal state passed by the encoder. The output sequence of the decoder is then passed to a Dense layer to process the output of the decoder. The training model takes the parameter sequence (encoder input) and the shifted output sequence (decoder input) as inputs and is trained to produce the unshifted output sequence (power profile). Both input and output training sequences must be padded to have the same length.

In inference, the input sequence can be padded but does not necessarily need to be. The output sequence can either be inferred until the model returns the first padding value or for a specified number of times (e.g. until the maximum length is reached). In the second case, a well-trained model returns padding values after the actual output sequence has been returned. Here, the second case is implemented for compatibility reasons with the other model architectures. To remove the teacher forcing, the encoder model is configured to take a parameter sequence as input and to return its internal state. This internal state is passed to the decoder as its initial state. When receiving an input, the decoder returns an output and its internal state. The output is initiated with the start token −2. In the inference loop, after every cycle, the internal state and

the last output are fed back into the decoder until the desired output sequence length (maximum process duration) is reached.

3.2.3 Parameter-Based Ensemble Learning

The ensemble model is a parameter-based ensemble model of several instances of the RNN LSTM architecture. The model is chosen based on the recipe ID, so each model specializes in the load profile forecasting of one process type only. This way, the most decisive parameter of the load profile shape does not have to be learned by the model anymore and the model can rather focus on the detailed parameter dependencies. The choice of the architecture of the single model is arbitrary, it just needs to be compared against its “single model” counterpart. The following listing presents the implementation of the parameter-based ensemble model:

```
# Input layer for all features
input_layer = Input(shape=input_shape, name='input_features')

# Split the input into recipe ID features and other features
recipe_id_features = input_layer[:, :, :self.number_of_recipe_ids]
other_features = input_layer[:, :, self.number_of_recipe_ids:]

# Create lists to store model layers for each recipe ID
model_layer_lstm1 = []
model_layer_dropout1 = []
model_layer_lstm2 = []
model_layer_dropout2 = []
model_layer_output = []

for i in range(self.number_of_recipe_ids):
    model_layer_lstm1.append(LSTM(64, return_sequences=True, name=f'lstm_layer1_recipe_id_{i}')(other_features))
    model_layer_dropout1.append(Dropout(0.2, name=f'dropout1_recipe_id_{i}')(model_layer_lstm1[i]))
    model_layer_lstm2.append(LSTM(64, return_sequences=False,
name=f'lstm_layer2_recipe_id_{i}')(model_layer_dropout1[i]))
    model_layer_dropout2.append(Dropout(0.2, name=f'dropout2_recipe_id_{i}')(model_layer_lstm2[i]))
    model_layer_output.append(Dense(output_shape[0], name=f'output_recipe_id_{i}')(model_layer_dropout2[i]))
    model_layer_output[i] = K.expand_dims(model_layer_output[i], axis=1)

# Concatenate all the model_layer_output tensors
concatenated_output = Concatenate(axis=1)(model_layer_output)

# Reshape recipe_id_features to match the shape of concatenated_output (for later multiplication)
recipe_id_features_reshaped = Reshape(target_shape=(self.number_of_recipe_ids, 1))(recipe_id_features)

# Use Multiply layer to perform element-wise multiplication
multiplied_output = Multiply()([concatenated_output, recipe_id_features_reshaped])

# Sum along the second axis to select the appropriate output
selected_output = K.sum(multiplied_output, axis=1)

model = Model(inputs=input_layer, outputs=selected_output)
model.compile(loss='mse', optimizer='adam')
```

The input layer receives the input data as a tensor with all features, including the one-hot encoded recipe ID. Then, the input data is split into two parts: recipe ID features (multiple features from encoding) and other features. Recipe ID features are used to identify the corresponding model instance, while other features are used for forecasting the load profile. In a for loop, lists are filled to store the recurrent and dense layers for each recipe ID. Layer names are assigned to simplify debugging. The model architecture for each recipe ID follows the RNN LSTM architecture described above, but only the non-recipe-ID features are passed to the input of the single models. Also, an additional layer after the final dense layer of each model expands the output tensors from each recipe ID along the first axis. This way, the list model_layer_output contains individual output tensors for each recipe ID, which can be concatenated in

the next layer to form one big tensor of the shape (number_of _recipe_ids, length_of _output_prof file) for the output of all models. The tensor with the recipe IDs is also reshaped, so both tensors can be multiplied. The recipe tensor is one-hot-encoded, i.e., the column of one categorical parameter is converted to one column for each value of this parameter, and the columns of the values are set to “0” or “1” depending on the categorical parameter value of the respective line. Due to the one-hot encoding, this multiplication isolates the output vector of the corresponding recipe ID, i.e., all other tensor elements are multiplied with zero. The summation along the first axis, therefore, produces the output vector of the corresponding recipe ID in the desired shape.

3.3 Confidence Estimator

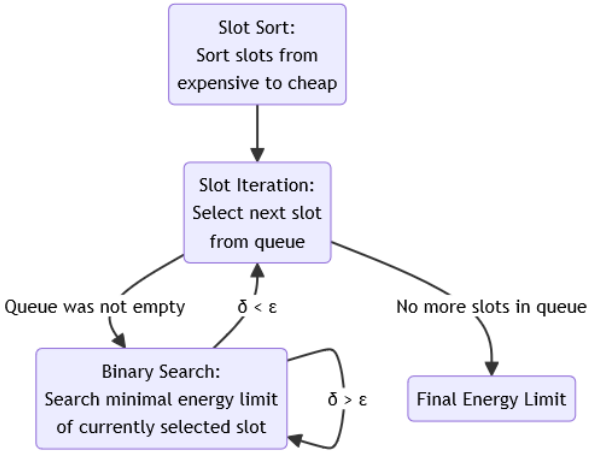
The selected approach encapsulates Manufacturing Execution Systems (MES) and treats them as black boxes. However, the inherent unpredictability of MES behavior from an external standpoint renders analytical optimization impractical. Consequently, we have chosen a rule-based approach to indirectly influence scheduling decisions.

3.3.1 Premises

The algorithm operates by optimizing energy costs through task scheduling, aiming to maximize energy consumption during periods of lower pricing while minimizing it during times of higher pricing. Ideally, all required energy would be consumed during the most economical time slot. This approach is particularly advantageous in scenarios where energy is not pre-ordered, as the algorithm's benefits are evident. However, even in cases where energy is pre-ordered, the algorithm enhances energy cost efficiency by reducing consumption during expensive time slots, thereby potentially increasing the profitability of energy sales.

3.3.2 Cost optimization

The CE-optimization algorithm influences the MES by transmitting energy limit profiles in the form of Virtual Energy Machines (VEMs), compelling the MES to schedule tasks that collectively consume either less or an equal amount of energy compared to the specified limit. There exist two viable approaches for the Confidence Estimator to impact the scheduling process. The first approach involves setting the energy limit of a time slot to zero initially and progressively increasing it for all spot market time slots, commencing with the cheaper slots until the MES can schedule all tasks. Conversely, the second approach starts with an "infinite" energy limit and iteratively adjusts the energy limits. In this approach, the algorithm begins with more expensive time slots and decreases the limit until the MES can no longer schedule all tasks. The algorithm's first approach, while effective, may initially impose narrow production time constraints due to the gradual increase in energy limits. This can inadvertently hinder the scheduler from generating a valid schedule, particularly for jobs spanning multiple time slots. The second approach circumvents this issue by refraining from imposing an initial energy constraint on the MES. Consequently, the algorithm adopts the more promising second approach, as depicted by the state machine and pseudo-code in Figure 5. Initially, a reasonable limit value is set for all time slots, such as the power capacity of the electrical grid connection. Subsequently, these limits are iteratively decreased for each time slot, starting with the more expensive slots. This iterative process continues until either the sum of the base and peak load is attained or until the minimum viable value is identified, which still permits the MES to generate a valid schedule.




```
foreach time slot t from expensive to cheap:
    e_limit[t] = base_line
    if tasks not schedulable:
        search_range = [base_line, line_limit]
        while diff(search_range) < epsilon:
            e_limit[t] = middle(search_range)
            if tasks are schedulable:
                search_range = lower_half(search_range)
            else:
                search_range = upper_half(search_range)
            end
        end
        e_limit[t] = upper_limit(search_range)
    end
end
```

Figure 5: CE-optimizer state machine algorithm

The Figures below illustrate the algorithms execution of 5 select moments from an exemplary optimization process to illustrate the impact of the algorithm steps on an energy profile. The x-axis in each graph represents the time slots, with a typical 1-hour resolution this would normally be 24 slots per day, we cut it down to 16 slots for space reasons. The green line represents the energy price for a given time slot. The red line is referred to as the line limit, it represents the maximum allowed energy consumption, normally based on physical properties of the electrical system, e.g. the energy limit of the connection to the grid. The blue-dotted line is the currently set energy limit, which is the subject of our optimization.

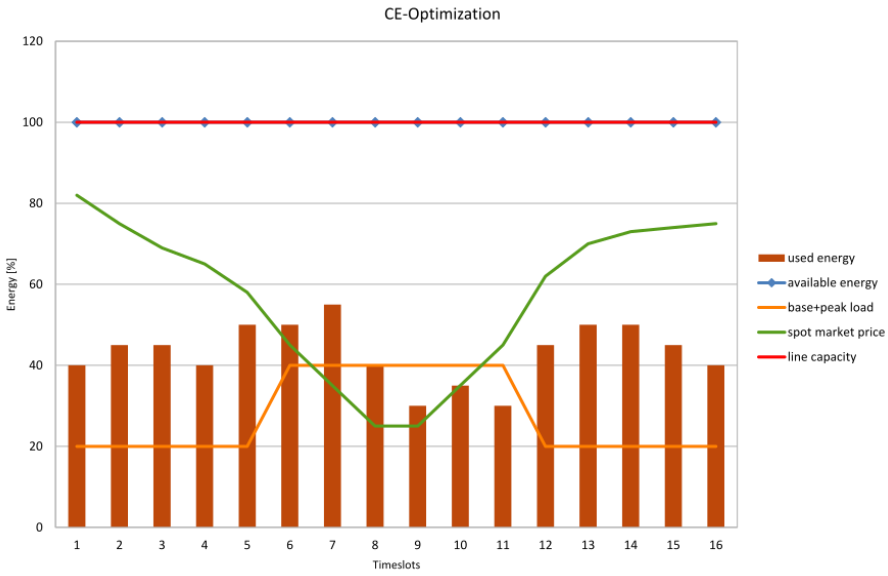


Figure 6: Initial setup for CE example optimization

Figure 6 shows the initial, unoptimized consumption profile with the blue line being fully blocked by the red line meaning that all slots are unrestricted.

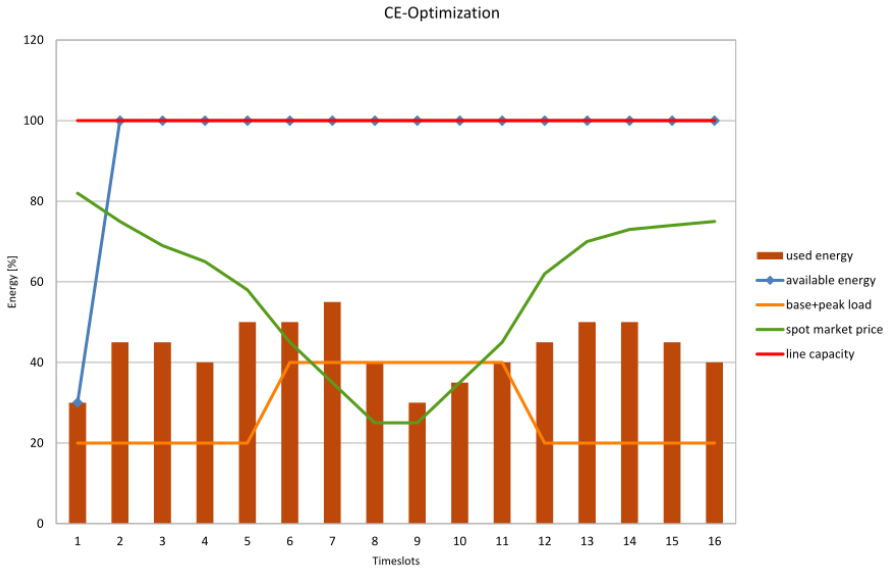


Figure 7: First step in CE-optimization process (minimizing the most expensive slot)

In step 1 as shown in Figure 7, we adjust the upper limit of the most expensive slot to the lowest possible value that still allows the underlying MES to schedule all required tasks within the given timeframe. The individual steps of finding this minimum value, using a binary search algorithm, are not shown in the graph. The final point the system settles on can be considered the minimum amount of energy needed at the most expensive time slot of the day. And since we assume that the total amount of energy consumption stays constant throughout the day we can now assume that the resulting schedule has either the same or, in almost all cases, a lower cost than the initial, unoptimized, schedule since in any other slot where the machine might now perform the displaced tasks, the energy cost will be lower.

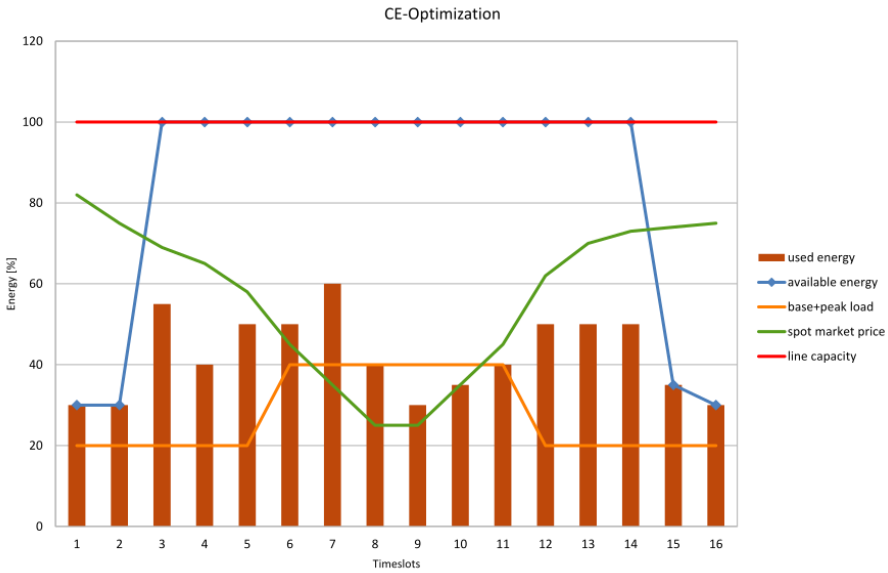


Figure 8: Step 4 in CE-optimization (minimized four slots)

Figure 8 shows the profile after we repeated this process three more times. The graph shows that the final, minimal points where the consumption limits settles does not need to be the same for all slots, the individual minimum value will strongly depend on the interdependencies of the production steps, the number of products to produce and the utilization of the factory.

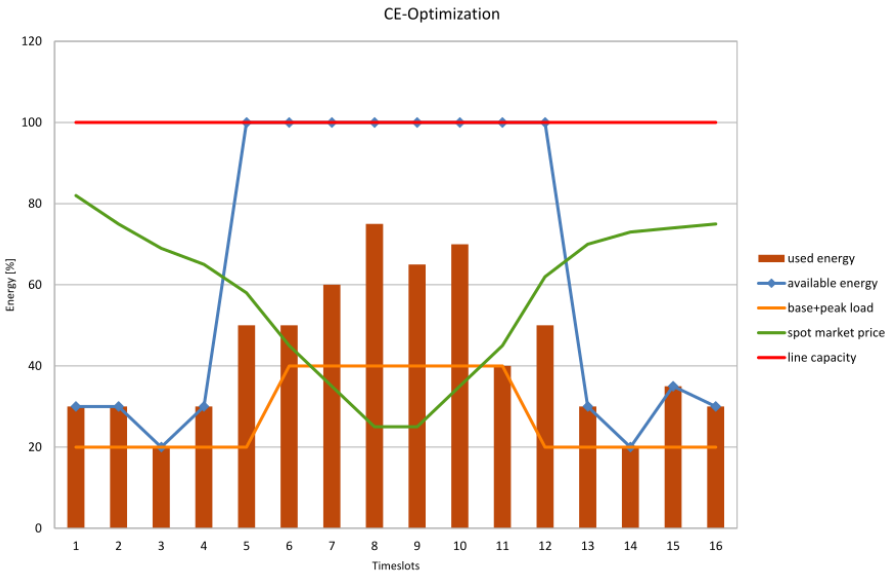


Figure 9: Step 8 in CE-optimization

Figure 9 shows the profile after 4 more steps of “pushing” energy consumption from the most expensive slots to the cheap slots. In our example we placed the expensive slots on the outside of the graph and the cheap slots in the middle to make the effect visually clearer. The final image (Figure 10) nicely shows how the amount energy that was “pushed out” of the expensive slots accumulates in the cheap, middle slots.

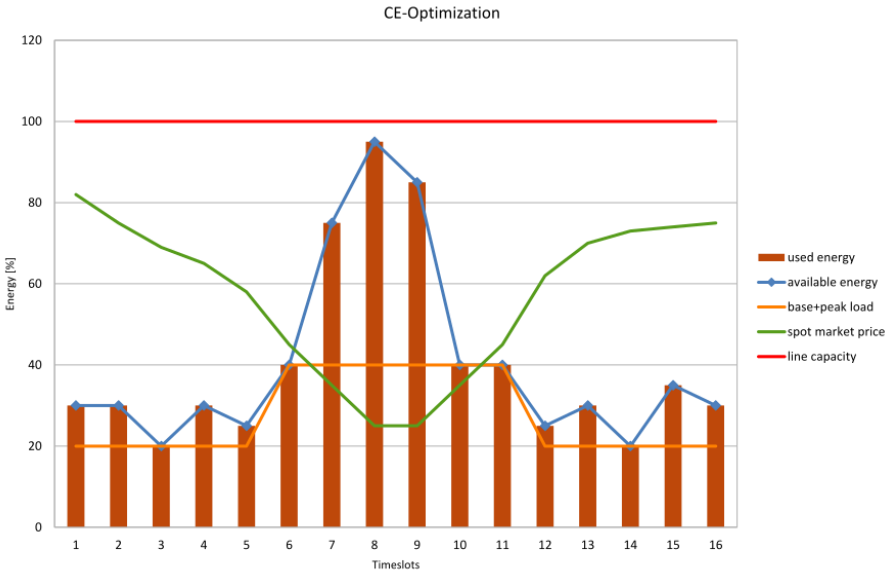


Figure 10: Final step in CE-optimization (all slots minimized, ordered by descending prices)

The fully optimized schedule is now providing the same amount of energy but takes advantage of the different energy prices during different time slots.

4 Implementation

Experimental verification of the optimization process was an essential part of the project. This section will give an overview on how we structured the implementation, how we achieved reliability in communication, experiment execution and evaluation and how we handled the computational strain of generating experiment results by enabling distributed execution of experiments.

4.1 General Setup

The setup process for the F4R experiments was complex. Each experiment includes the simulation of multiple production lines with complex machines, integrated with real and synthesized energy price profiles, a complex and configurable custom MES we developed and one or more of various available energy consumption predictors, that need to fir the production line setup used in the experiment.
To handle this complexity in configuration we split the configuration setup into different directories – see Figure 11 for an overview. The central aspect for experiment specification is the setups directory. This directory contains a sub-directory for each available scenario. Note that the scenarios are, conceptionally grouped into different iterations, e.g. There are currently 15 individual variations of Scenario 2, which all have a different designation, all starting with SC 2. This conceptional clustering of scenarios is not represented in directories but is only visible through the scenario names.
The scenario directories contain different files that specify parameters for all the simulated aspects of an experiment. It was important to maintain each scenario as an enclosed element that contains all the information it needs with as little reference to other directories as possible. Therefore, elements that are shared between scenarios, for example specific production line setups, are still repeated in each scenario directory. A typical reason for such shared data are the scenario variations, which mean that a lot of configuration information will overlap and only certain specific aspects of the scenario, e. g. the price profile, will vary. The decision to repeat information in such a manner introduces the risk of inconsistencies within a scenario and to tackle this problem we defined the scenario directories as “generate able” directories, meaning that we use scripts to generate the directories based in information, currently hardcoded in the generation scripts. This approach combines the readability of well formed configuration files, as opposed to having to look through configuration scripts, with the “packed” structure that allows to unambiguously identify which configurations belong together and makes it easy to package these configurations together with their results to make sure that all outputs can always be linked to the configurations they were produced with.

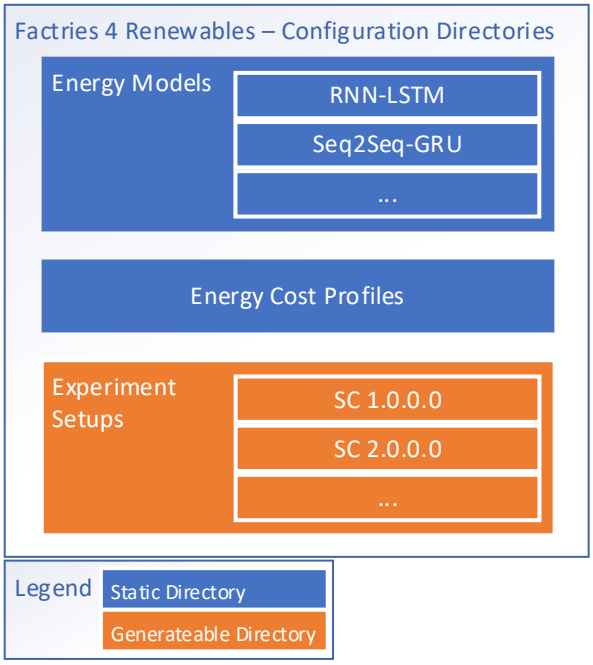


Figure 11: Configuration directories

The figure shows that the “Energy Models” and “Energy Cost Profiles” directories are considered static directories, meaning that they are checked into the git repository and tagged specifically and will not be re-generated before each experiment run. The main reasons for this were that the cost profiles change very rarely and the energy models need to be trained, which, depending on the model architecture, can take a very long time.

4.2 Simulation Runs

A dedicated, central Simulator component controls the simulation. The purpose of that component is to provide a central entry point for the simulation, as well as a central funnel for the different configurations, and to set up the simulation workflow by connecting all the other components used in the simulation. The configuration aspect of the Simulator includes preparing and specifying the debug and output directories for the created components (if applicable), this makes sure that all debug outputs and results that belong to a single simulation are properly put together for later inspection and offline analysis/debug. After the setup is finished, the simulation loop is driven by the Confidence Estimator and the simulator. After the active parts of the simulation are finished, i. e. when the Confidence Estimator main loop returns, the Simulator component springs into action again for some additional debug logging and some minor checks of the results, intended purely for development purposes. The simulation results themselves are stored in the prepared filesystem structure (as provided by the Simulator component during configuration) by the Confidence Estimator.

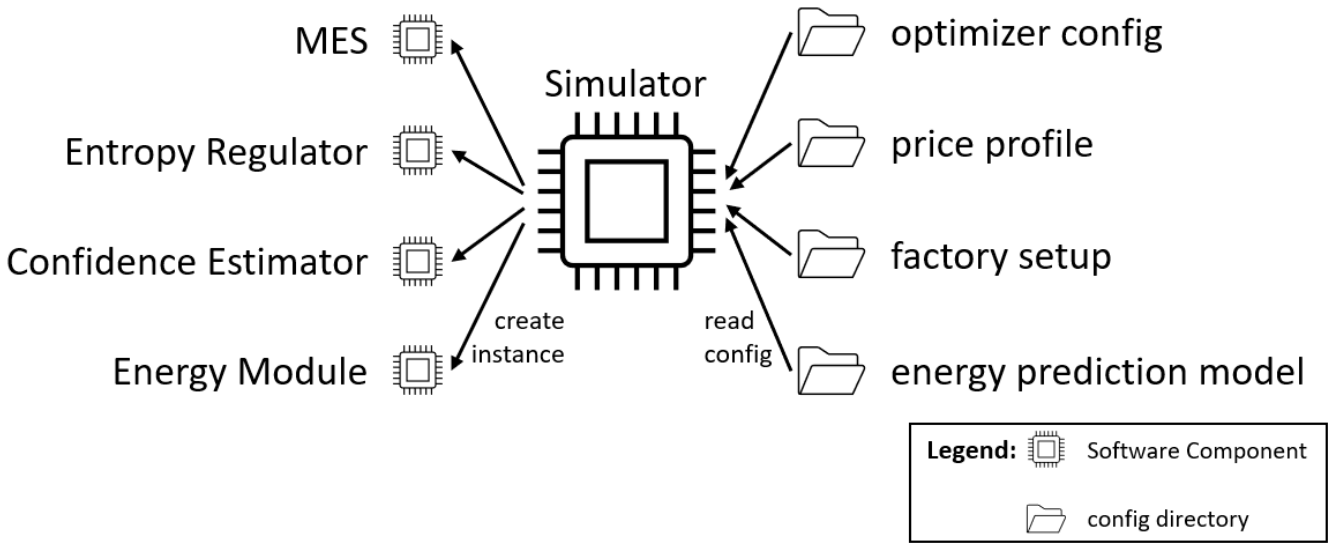


Figure 12: Experiment configuration overview

4.3 Output Collection

Given the complexity of the simulation we expected a large amount of output data to be generated, especially during development when additional debug outputs will be required. Our main goal in handling this data was to ensure consistency and completeness. It is paramount for us to be able to check each set of outputs to ensure that the data was generated with the correct inputs and that the aspects of simulation that overlap with other scenarios or variations within the same scenario, are consistent and therefore allow comparison between the different result sets. We achieved this by generating distinct experiment directories for each run of each scenario within each experiment. This led to the output directory structure outlined in Figure 13.

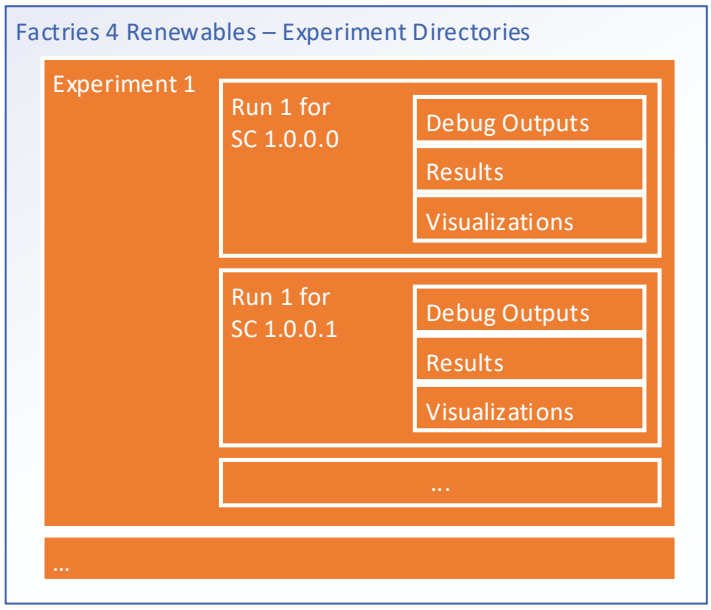


Figure 13: Experiment directory structure

The figure uses the same color scheme as Figure 12, meaning that all the directories are generated. Each experiment in our project contains one or more run and each run simulates one or more scenarios. The option to make various runs per experiment was introduced in case we introduce a random element into the simulation, where having multiple runs of the same setup would make sense to provide results that are robust against outliers in their random elements – our current setups do not include such elements and therefore only require a single per experiment.
Within each simulated scenario in our experiment we generate output directories for any debug output files generated during the experiment, one output directory for the final results and a third directory containing any additional visualizations that could be generated during the experiment.
Please note that directly after experimentation the results in the output directories are contain the output data of the experiments, not yet the KPIs and final evaluation. The comparable results are generated in a separate offline step using specific scripts. This disconnect was deliberately created to allow easier change of evaluation scripts, without having to re-run experiments and to enable the post-simulation introduction of potential post-processing steps. Both aspects aim to minimize the need to re-run simulations as we soon realized that each simulation requires a significant amount of processing power.

4.4 Post-Processing and Evaluation

In a final step we use dedicated evaluation scripts to generate KPIs and put them into context, e. g. by comparing them, for each scenario in an experiment. These scripts collect the step-by-step data from the debug directories and the final schedule from the output directory, in combination with setup data used in the experiment to compare the impact of our optimization algorithm on the total energy cost of production for the different scenarios. The mathematical details on the KPIs we calculate are explained at the beginning of the results section.
The evaluation is split into different scripts, the first evaluation scripts calculates the KPIs for all scenarios in a experiment and creates a tab-separated-values (.tsv) file where each line is a scenario and a high number of columns represent all input variables and values that are relevant for further evaluation, comparison and visualization. The next script groups the lines of the table by various criteria to allow a comparison of KPIs based on utilization rate, price profile type, pricing scheme (i. e. specific combination of spot market purchases, local PV and fixed price energy futures). And an optional final script is used to generated visualizations for the comparative results, mostly as various box plots with some additional overlays, as well as Latex scripts that can be used directly within a Latex interpreter to provide beautified tables that show the relevant information obtained via evaluation and comparison – intended to be used in dissemination works.

5 MES

A Manufacturing Execution System (MES) is a software-based system used in manufacturing environments. It serves as a bridge between the enterprise resource planning (ERP) systems and the shop floor, providing real-time visibility and control over the manufacturing process.

5.1 Key Components

The key components and functions of an MES in general:

- Production Planning and Scheduling: MES helps in creating and optimizing production schedules based on factors such as order priorities, resource availability, and production capacity.
- Resource Allocation and Management: MES assigns resources such as equipment, materials, and labor to specific tasks or production orders efficiently, ensuring optimal utilization of resources.
- Work Order Management: It manages and tracks work orders throughout the production process, from creation to completion, ensuring that production tasks are executed according to plan.
- Inventory Management: MES tracks inventory levels in real-time, including raw materials, work-in-progress (WIP), and finished goods. It helps in minimizing inventory costs and maintaining optimal inventory levels.
- Quality Management: MES monitors and controls quality throughout the manufacturing process by enforcing quality standards, conducting inspections, and capturing data related to quality issues or defects.
- Data Collection and Analysis: It collects data from various sources such as machines, sensors, and operators on the shop floor. MES analyzes this data to provide insights into production performance, efficiency, and quality metrics.
- Real-time Monitoring and Control: MES provides real-time visibility into production activities, allowing supervisors and managers to monitor progress, identify bottlenecks, and take corrective actions as needed.
- Traceability and Compliance: MES enables traceability by tracking the genealogy of products and materials throughout the production process. It also helps in ensuring compliance with regulatory requirements and industry standards.
- Integration with Other Systems: MES integrates with other enterprise systems such as ERP, supply chain management (SCM), and product lifecycle management (PLM) systems to facilitate seamless data exchange and synchronization across the organization.

Overall, Manufacturing Execution Systems play a crucial role in improving operational efficiency, reducing costs, enhancing quality, and enabling agility in manufacturing processes.

Note on available open source software:

The https://www.capterra.com/manufacturing-execution-software/ link provides a comprehensive list of available Manufacturing Execution Software systems. There is plenty of open-source software that provides ERP systems and MES. But availability doesn't mean it is easy to use.

5.2 MES System Architecture

This section explains the basic requirements of a Manufacturing Execution System (MES) regarding the F4R project. The following Figure depicts potential inputs for MES. The Basic information box represents the information used during the planning and is somehow fixed. The Bill of Materials (BOM) and recipe are the most apparent primary data. Some advanced topics like production line layout, and alternative materials and processes are ignored here.

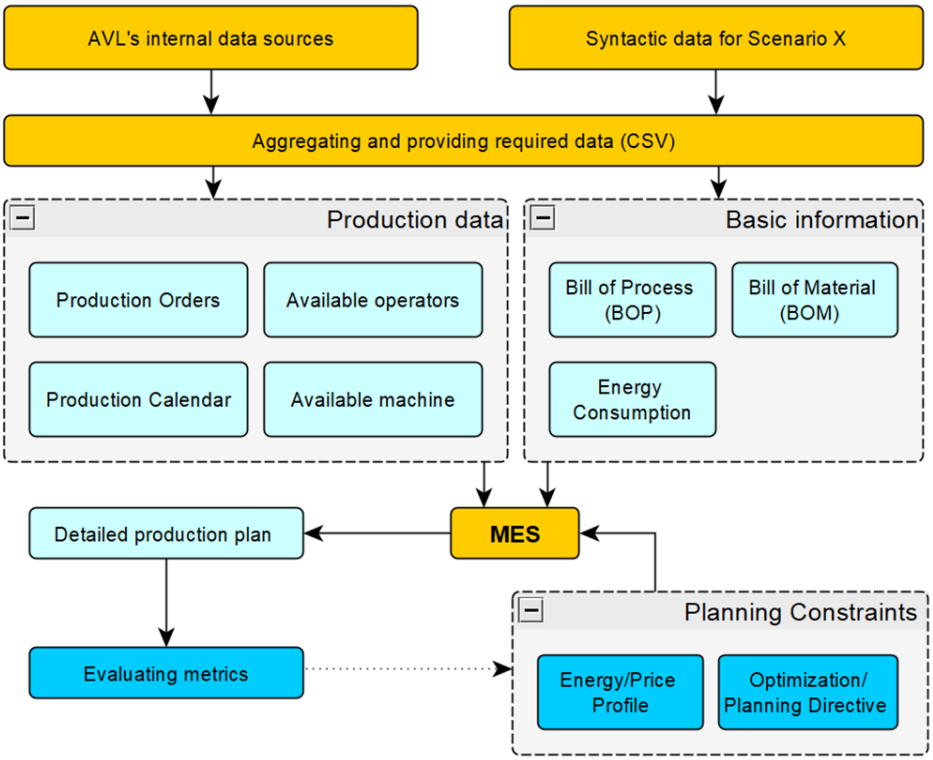


Figure 14: MES Inputs

5.2.1 Production Planning Basic Terms

MES / Scheduler

The main job of MES is scheduling and allocating resources for each task. It defines the start time, end time, production equipment, and operators/workers for each task. Production orders drive the task list.

BOM (Bill of Materials)

"A BOM is a comprehensive inventory of the raw materials, assemblies, subassemblies, parts, and components, as well as the quantities of each needed to manufacture a product. In a nutshell, it is the complete list of all the items that are required to build a product." [2]

In this project, we assume that all required materials are available and thus ignore the BOM.

Production Order

When planning production, different approaches can be taken to consider production orders. For this document, production orders are defined as a list of products to be manufactured by a certain deadline.

A production order item defines the name and amount of the required product in a predefined period. The following example shows that ten pieces of "Battery_A", and 5 pieces of "Battery_B" are supposed to be scheduled in 200 time units (e.g., minutes). Some other directive fields like priority also can be added here. The extra fields in different versions may differ slightly.

Machines and Operators

For production planning, necessary types of equipment are defined in advance. We assumed each process is associated with one machine and one operator. We can assign more than one machine by defining another (sub)process. In other words, instead of defining a process with two machines, we define two sub-processes with one machine each. If a task requires no machine, we can assign it to a dummy/void machine, in the same way for the operators.

Production Calendar

It defines the available days/hours for production. It can be considered as a separate object, or it can be integrated with the Energy Profile.

Recipe

Production Recipe: defines the sequence of required tasks for each specific product. Necessary production equipment, raw materials, and operators are also represented in a Production Recipe.

The input parameters of the MES class are Production Orders, Machines, Operators, Recipes, and VMs. Although the VMs are from Machine type, for the sake of clarity and performance they are separated. All parameters are provided in the form of pandas.DataFrames.

5.2.2 MES Standards

ANSI/ISA-95, more commonly referred to as ISA-95, is an international standard for enterprise and control systems integration developed for manufacturers.

The ISA-95 standard was jointly developed by the International Society of Automation (ISA), formerly known as the Instrumentation, Systems, and Automation Society, and the American National Standards Institute (ANSI) to provide abstract models and standard terminologies for the exchange of information between the enterprise business systems and manufacturing operations systems in an enterprise.[3]

5.2.3 MES Implementation for Factories for Renewables

This section is a user manual that provides step-by-step instructions for configuring and using the MES software. It is a supplementary document to the Python source code, and it is not intended to be a textbook about the MES systems. It explains necessary points regarding the usage of the developed MES. The explanation is based on the "rapidScheduler.py" script, which is a modified version of "scheduler.py". The

[2] www.techtarget.com
[3] https://www.techtarget.com/searcherp/definition/ANSI-ISA-95

modified version works faster and has some interface differences. The "rapidScheduler.py" version is used as default scheduler in the experiments and results.

5.2.3.1 Input data (Logical Interface)

The MES planning interface relies heavily on the MES definition and implementation. The following primary input data are necessary for planning.

- Production Orders
- BOM
- Recipe/BOP
- Production Machine
- Operators
- Production Calendar
- Planning and Optimization Directives

In this project, we also require the following inputs:

- Energy Profile
- Energy Consumption patterns of processes

Certain items on this list remain constant and are subject to rare changes (e.g., Recipe/BOM).

5.2.3.2 Input Format and Preparation

We used Python data types for data exchanges in the planning system. The rapidScheduler.py file contains a class called rapidMES. This class is initialized with 5 pandas.DataFrame instances.

5.2.3.2.1 orders_df

```
orders = [
     [1, '50_kWh_BS5', 10, 0,  20 ],
     [2, '50_kWh_BS5', 10, 5,  25 ],
     [3, '50_kWh_BS5', 10, 10, 30 ],
     [3, '50_kWh_BS5', 10, 15, 30 ]
 ]

orders_cols = ['order_ID', 'product_name', 'product_amount', 'product_start_time', 'product_deadline']

orders_df = pd.DataFrame(orders, columns=orders_cols)
```

5.2.3.2.2 machines_df

The function Utilities.getMachineList() provides a sample DataFrame for machines that is valid and can be used as a reference.

```
def getMachineList():
    ma_cols = ['machine_idx', 'machine_Type', 'available_machine']

    ml = [[0, 'stacking_robot', 40],
          [1, 'welding_robot', 40],
          [2, 'gluing_robot', 40],
          [3, 'packing_robot', 40],
          [4, 'testing_station', 40],
          [5, 'trolley', 55],
          [6, 'stacking_robot2', 40],
          [7, 'welding_robot2', 40],
          [8, 'gluing_robot2', 40],
          [9, 'packing_robot2', 40],
          [10, 'testing_station2', 40],
```

```
          [11, 'trolley2', 55],
          [12, 'wind', 100]]

    ml_df = pd.DataFrame(ml, columns=ma_cols)
    return ml_df
```

5.2.3.2.3 operators_df

The method Utilities.getOperatorList() can be used to obtain a list of operators.

```
def getOperatorList():
    op_cols = ['operator_idx', 'operator_Type', 'available_operator']

    ol = [[0, 'SkilledWorker', 75],
          [1, 'Transporter', 75],
          [2, 'SkilledWorker2', 75],
          [3, 'Transporter2', 75],
          [4, 'Nobody', 95]]

    ol_df = pd.DataFrame(ol, columns=op_cols)
    return ol_df
```

5.2.3.2.4 rcp

```
 rcp_cols = ['rcp_id', 'product_name', 'sequence', 'process_name', 'machine_type', 'operator_type', 'duration',
'energy_consumption_curve', 'prerequisites']

ecc = [2] * 40
batchSize = 10
batch_rcp = [[0, '50_kWh_BS5', 0, 'stacking', 'stacking_robot', 'SkilledWorker', 2, ecc, []],
             [1, '50_kWh_BS5', 1, 'stacking_to_gluing', 'trolley', 'Transporter', 1, ecc, [0]],
             [2, '50_kWh_BS5', 2, 'gluing', 'gluing_robot', 'SkilledWorker', 2, ecc, [1]],
             [3, '50_kWh_BS5', 3, 'gluing_to_welding', 'trolley', 'Transporter', 1, ecc, [2]],
             [4, '50_kWh_BS5', 4, 'welding', 'welding_robot', 'SkilledWorker', 3, ecc, [3]],
             [5, '50_kWh_BS5', 5, 'welding_to_packing', 'trolley', 'Transporter', 1, ecc, [4]],
             [6, '50_kWh_BS5', 6, 'packing', 'packing_robot', 'SkilledWorker', 2, ecc, [5]],
             [7, '50_kWh_BS5', 7, 'packing_to_quality', 'trolley', 'Transporter', 1, ecc, [6]],
             [8, '50_kWh_BS5', 8, 'quality', 'testing_station', 'SkilledWorker', 1, ecc, [7]]]

batch_rcp_df = pd.DataFrame(batch_rcp, columns=rcp_cols)
```

In the next step, it is necessary to update the RCP with an energy curve, which contains the average consumption of each task. It is provided by EM, and it can be a curve or constant value.

```
# update the RCP with energy curve.
for idx, row in batch_rcp_df.iterrows():
    r_id = row['rcp_id']
    r_dd = row['duration']
    slotBasedCurve = []
    for idx2, row2 in ecc_df.iterrows():
        if row2[0] == r_id:
            ls = eval(row2['mean'])
            # print('\n id:',r_id, len(ls), mean(ls))
            for i in range(round(len(ls) / 15)):
                sub_ls = ls[i * 15:(i + 1) * 15 - 1]
                x = round(mean(sub_ls))
                slotBasedCurve.append(x)
                batch_rcp_df.at[idx, 'energy_consumption_curve'] = slotBasedCurve
```

5.2.3.2.5 *eVM* is created using the 'getNewEnergyProfile()' function.

```
def getNewEnergyProfile(start_Time, End_Time):
    # hourly curves
    # [lower_limit  upper_limit  priority] one day
    newEP = [[1000, 3800, 1],  # 1
             [1000, 3700, 1],
             [4115, 3625, 1],
             [3115, 3525, 1],
             [3115, 3425, 1],
             [3110, 3315, 1],
             [3110, 3215, 1],
             [4110, 3215, 1],
             [4110, 3115, 1],
             [4110, 3015, 1],  # 10
             [4110, 2915, 1],
             [5015, 2825, 1],
             [6000, 2700, 1],
             [115, 225, 1],
             [110, 215, 1],
             [110, 215, 1],
             [110, 215, 1],
             [110, 215, 1],
             [115, 225, 1],
             [115, 225, 1],  # 20
             [115, 225, 1],
             [110, 215, 1],
             [110, 215, 1],
             [110, 215, 1]]  # 24 one day

    # convert to 15 minutes
    fine_grain_ep = []
    for desiered_days in range(10):
        for oneline in newEP:
            for step in range(4):
                fine_grain_ep.append(oneline)

    return pd.DataFrame(fine_grain_ep[start_Time: End_Time], columns=['lower_limit', 'upper_limit', 'priority'])
```

5.2.3.3 Instantiation and Execution

All the required input DataFrames are prepared and now we can proceed with instantiating rapidMES. Although there are many methods in the rapidMES class, the most important one is .plan() which performs the job. Depending on the success or failure, appropriate messages are displayed, and the results are logged in related files and/or displayed on the screen.

```
mymes = rapidMES(orders_df, machines, operators, batch_rcp_df, eVM)
major, minor = mymes.getVersionInfo()
print(f'Version {major}.{minor}')

mymes.processExplosion()
res = mymes.plan()

if not res:
    print("Scheduling failed.")
```

6 Project Results

The goal of our project was optimizing the use of green energy during the production process by shifting energy intensive production steps to times of more renewable energy in the energy mix. We then simplified the criteria of green-energy-availability to energy price, under the assumption that lower prices signify more renewable energy available. This chapter will now summarize the results we achieved with our algorithm in experiments, based on realistic scenarios, towards this end. However, our project consists of several components that all contribute to the final results, but some of them offer scientifically interesting output even outside of the context of our final goal, first and foremost the Energy Module component that uses modern Machine Learning methods to estimate the energy consumption of machines under specific parameterizations, based on history measurements. Following this, the results section will be split into two parts: first we present and discuss the results obtained for prediction energy consumption from various machine learning approaches, based on different data sources. In a second subsection we then discuss the final results of energy-price aware schedule optimization. The correct, reliable and accurate prediction of energy consumption is essential to achieve these results, as they are the only way to properly compare different schedule candidates. However, during experimentation we deliberately de-coupled the performance of the schedule optimization and consumption prediction, by ensuring that the prediction itself was always accurate, thereby avoiding issues where the impact of optimization, positive or negative, is lost within the distortion created by some sub-par prediction. Each of the two aspects of results will present their own findings and draw their own conclusions for their respective context.

6.1 Energy Consumption Prediction

The different model architectures used in energy consumption prediction are tested on synthetic energy data based on data from the Battery Innovation Center (BIC) at AVL in Graz. The number of total samples is set to 1000. The dataset is split into training, validation, and test datasets with a ratio of 70/20/10%. the number of training and validation samples was multiplied by a factor of 5 by adding Gaussian noise where $\mu = 0$ W and $\sigma = 5.0$ W.

The distribution of recipe IDs in the respective datasets is shown in Figure 15. As can be seen from the figure, the share of recipe IDs is not equal across the datasets for all recipe IDs (although for most recipe IDs it roughly is). However, this does not impair the validity of the split since also, in reality, the share of executed recipe IDs might vary over the lifetime of a factory, which is the same as a different distribution within the training/validation dataset and the test dataset.

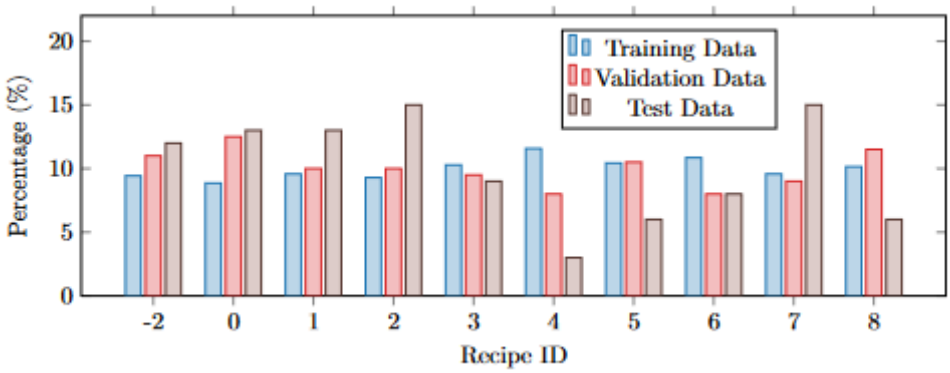


Figure 15: Percentage of recipe IDs in the training, validation, and test datasets.

Figure 16 shows the training history of the Ensemble LSTM architecture as an example. The training history includes the training loss as Mean Squared Error (MSE), the validation loss (MSE), and the learning rate for each epoch of the training on logarithmic scales. The cyclical learning rate pattern that oscillates between the minimum and the decreasing maximum is clearly visible. The training loss value decreases more monotonically than the validation loss value. Starting around epoch 35, the graph shows that the training reaches the state of overfitting, i.e., the model performs significantly better on the training dataset than on the validation dataset. It can also be seen that the loss values of both training and validation often increase temporarily with an increasing learning rate, but perhaps to the benefit of escaping a local minimum.

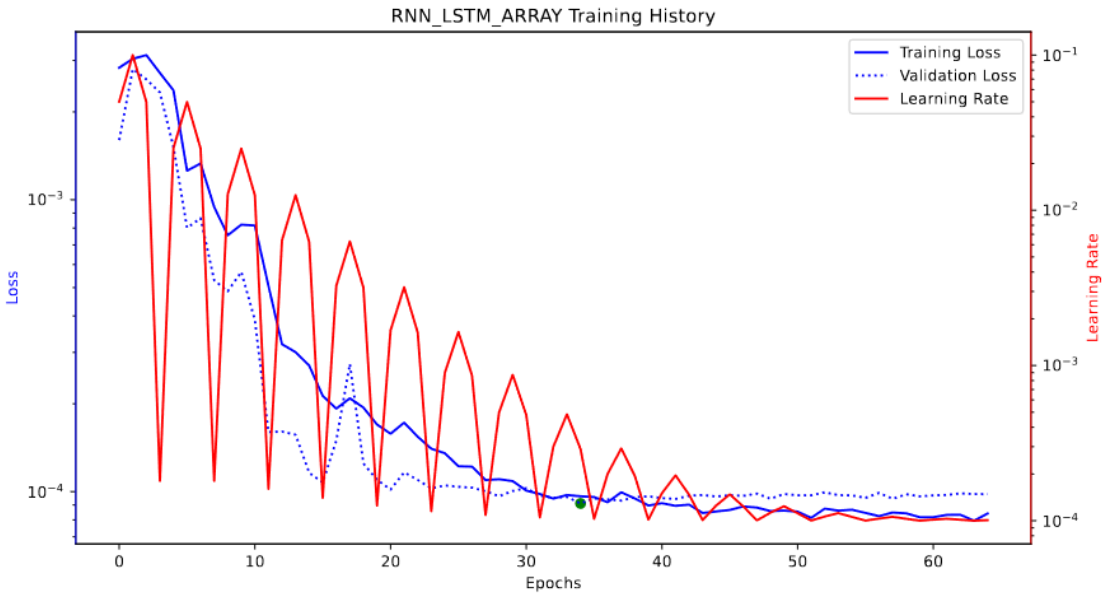


Figure 16: Training history of the Ensemble LSTM model.

The training results and parameters that form the basis for comparing the different model architectures are three-fold:

- MSE Comparison: The MSEs with and without padding values are shown in Figure 17.
- Training Durations: The training durations are shown in Figure 18.

- Inference Duration Comparison: Finally, the inference duration is compared in Figure 19.

The comparison of MSEs shows that the RNN and Seq2Seq architectures are pretty much on par, with the Seq2Seq architecture performing slightly worse than the others. The ensemble learning LSTM architecture based on the recipe ID performs best, its prediction MSE is better than the single RNN architectures by a factor of almost 10. Its MSE of 345.1 W^2 compared to the mean power value of 558.8 W means that it achieves an average prediction error of approximately 19 W, which is substantially lower than the mean power itself (3.3 %). The NN makes by far the worst predictions; the predictions from the test set reveal that the predictions by the NN are all identical, independently of the input parameters. A closer inspection of the consistent prediction vector indicates that it is the average of all power profiles from the training dataset, i.e., the NN does not learn the underlying relationships in the training data. This can be caused by an insufficient model complexity (e.g., too few layers or neurons) or by an inappropriate architecture. Since the MSE also does not improve with up to 40 layers, it can be concluded that the NN is not appropriate for predicting power profiles just from process parameters. The NN architecture has the strategic disadvantage to the RNN architectures in that it does not model any relationship between the output values, so each power value has to be determined just based on the input parameters which leads to very complex functions.

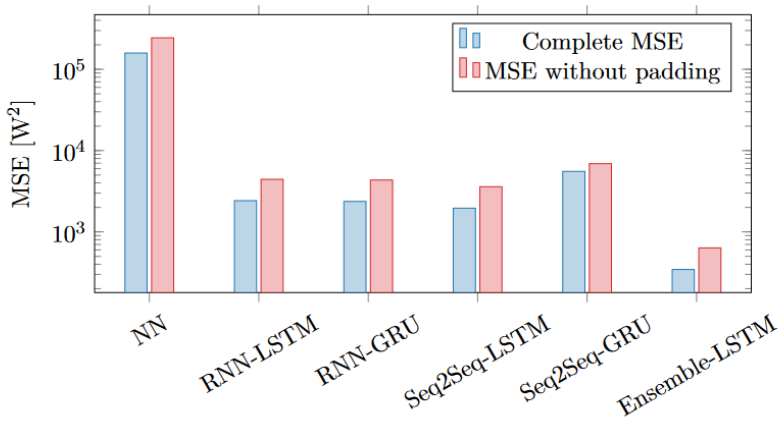


Figure 17: MSE of the different model architectures.

Comparing the values of the prediction MSE with and without padding values in Figure 17 shows that the complete MSE, including padding values, is slightly better for all architectures, i.e., the padding values are predicted better than the power values. This is not surprising since the padding values are the most regular values in the power vectors. The factor between the MSE with and without padding values is roughly 2 for most architectures; for the NN, it is roughly 1.5, and for the Seq2Seq model with GRUs, it is roughly 1.2.

The training durations (see Figure 18) vary between 23.5 s for the NN and 939.1 s for the ensemble learning architecture. There is no clear relation between training duration and accuracy: the accuracy of the RNN and Seq2Seq architectures are similar, while their training durations differ by a factor of roughly 10. The training duration seems to depend more on the model architecture. Likewise for the inference durations (see Figure 19), while most models' inference durations are around 0.1 s or below, the inference durations of the Seq2Seq models are around 2 s. The longer inference duration of the Seq2Seq

architectures can be explained by the recurrent nature of the inference process with encoder and decoder, and the less efficient inference with explicit Python code than with the efficient Tensorflow backend.

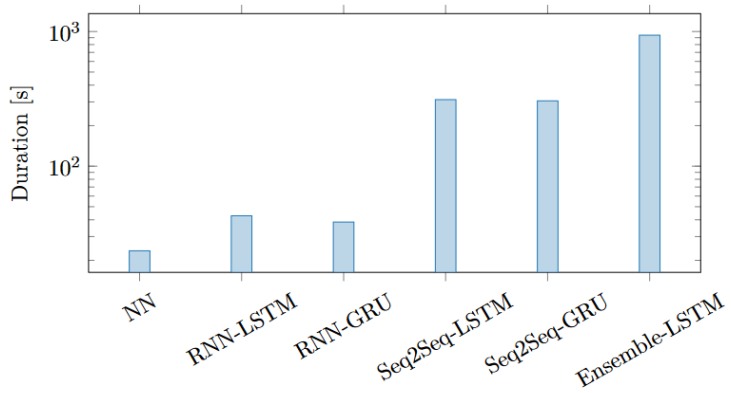


Figure 18: Training durations of the different model architectures.

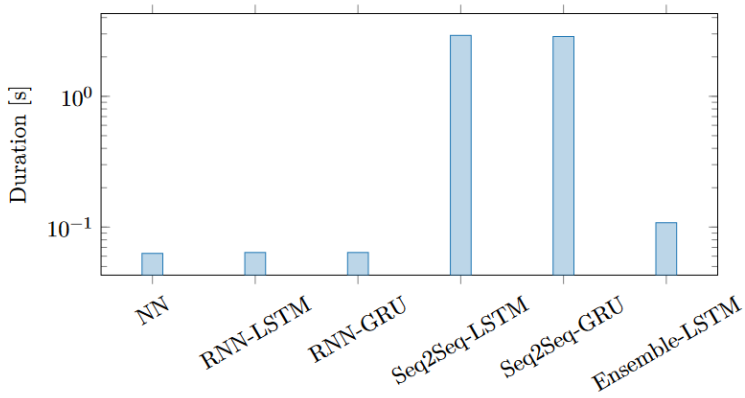


Figure 19: Inference durations of the different model architectures.

In conclusion, the ensemble learning architecture with models for each recipe ID seems to be most appropriate for use cases where the recipe ID is the most decisive parameter for the power profile, while the NN architecture is utterly inappropriate.

6.2 Energy Aware Schedule Optimization

6.2.1 Evaluation Metrics

Our investigation is aimed at demonstrating the enhancements in energy efficiency resulting from the integration of price awareness into the scheduling process. The experiments will thus concentrate on assessing the transition from price-unaware to price-aware scheduling across various application scenarios, factory utilization rates, and energy cost configurations. We adopt both technical and financial perspectives to evaluate performance enhancements. From a technical viewpoint, we emphasize the capacity to shift loads to periods with lower energy prices along the price curve. From a financial viewpoint, we analyze how these load shifts interact with energy prices and contracts to ascertain potential monetary savings. Both perspectives involve a metric based on the ratio between the costs incurred by unoptimized schedules and those optimized, calculated by aggregating weighted liabilities from future contracts with the sum of spot market transactions, as shown in Equation 1.

Equation 5: cost function

$$c_{total} = \left(E_{future} \cdot p_{future}\right) + \sum_{r=6}^{22} \left(E_r \cdot w_r\right),$$

The cost is represented by c, E denotes the utilized energy, p represents the price, and w is a weighting factor. The subscript $future$ indicates values derived from pre-negotiated energy contracts, while the subscript r serves as the running index for operation hours. The factory's operational timeframe spans from 6:00 to 22:00. The disparity between the technical and monetary perspectives lies in the treatment of contracts and the values assigned to the weight variables w. From a technical standpoint, E_{future} and p_{future} are presumed to be zero, and the weight is determined by the rank of each price in the price curve. For instance, a price curve with values {0.14, 0.12, 0.32} would be translated into ranks {1, 0, 2}. By utilizing ranks as weights, the influence of energy prices is normalized to the shape of the price curve, resulting in a summation of rank-weighted energy consumption shifts. This metric is applied to both unoptimized and optimized schedules to quantify the amount of energy shifted to cheaper timeslots, weighted by the distance it was shifted in ranks. Conversely, from the monetary perspective, actual contracted values for E_{future} and p_{future} are utilized, with the actual prices serving as weights w. In this metric, the disparity between unoptimized and optimized values reflects the real change in energy cost for the given scenario. Both metrics are calculated in both absolute and relative forms. The relative improvement is computed as 1 minus the total optimized cost divided by the total unoptimized cost. Additionally, alongside the main metrics, machine utilization and price profile characteristics are computed for each scenario. Machine utilization is defined as the duration a machine spends in active production divided by the total available production time.

6.2.2 Usecase Description

With the Factories 4 Renewables project we evaluated various scenarios. Three base scenarios were solidified into stable scenario setups and one of those three scenarios was used to illustrate the performance of our algorithm. The other two scenarios can be used to explore and investigate special application cases.

Our most fundamental scenario was scenario 1 (SC 1) which we titled sand-production due to single step, single machine products defined in this scenario. The reason for this simplification is to reduce the scheduling complexity for the MES so we can focus on the optimization process of the MES. Note: the simplicity in single step, single machine production comes from the fact that these products have no dependencies, are finished in a single step and can therefore by scheduled at any position of the schedule, bar any other additional restrictions, like limited machine availability.

The next scenario, entitle scenario 2 (SC 2) is our main evaluation scenario. It is based on the Battery Innovation Center (BIC) at AVL, in Graz and uses differently sized batteries and products. The energy consumption profiles are based on actual measurements taken from the production robots at the actual BIC-site.

The last scenario we fully designed, scenario 3 (SC 3), expands scenario 2 by including PV production and battery storage. However, our preliminary tests showed that the impact of PV and batteries was limited to the rare cases where an asymmetry in buying and selling prices for spot-market energy collides with especially small price changes and a lack of energy futures. Given the limited project time available we

decided not to produce additional results for these specific scenarios, but still provide the setups in case we decide to run these iterations sometime in the future.

Table 2: Scenario variations of type 1 and 2

Name	Utilization	Price Curve
SC 1.0.0.0	High	Type0
SC 1.0.1.0	High	Type1
SC 1.0.2.0	High	Type2
SC 1.0.3.0	High	Type3
SC 1.0.4.0	High	Type4
SC 1.1.0.0	Low	Type0
SC 1.1.1.0	Low	Type1
SC 1.1.2.0	Low	Type2
SC 1.1.3.0	Low	Type3
SC 1.1.4.0	Low	Type4
SC 1.2.0.0	Medium	Type0
SC 1.2.1.0	Medium	Type1
SC 1.2.2.0	Medium	Type2
SC 1.2.3.0	Medium	Type3
SC 1.2.4.0	Medium	Type4

Name	Utilization	Price Curve
SC 2.0.0.0	High	Type0
SC 2.0.1.0	High	Type1
SC 2.0.2.0	High	Type2
SC 2.0.3.0	High	Type3
SC 2.0.4.0	High	Type4
SC 2.1.0.0	Low	Type0
SC 2.1.1.0	Low	Type1
SC 2.1.2.0	Low	Type2
SC 2.1.3.0	Low	Type3
SC 2.1.4.0	Low	Type4
SC 2.2.0.0	Medium	Type0
SC 2.2.1.0	Medium	Type1
SC 2.2.2.0	Medium	Type2
SC 2.2.3.0	Medium	Type3
SC 2.2.4.0	Medium	Type4

The variations for Scenario 1 and Scenario 2 are identical as those two scenarios only differ in the product specifics.

Scenario 3 introduces two additional parameters: PV Peak production (as percentage of the total required energy for production divided on a typical PV production profile) and Transport Fee – where Transport Fee is currently set to a fixed rate and not varied.

Table 3: Scenario variations of type 3

Name	Utilization	PriceCurve	PV Peak	Transport Fee
SC 3.0.0.0	High	Type0	None	0.5
SC 3.0.0.1	High	Type0	High	0.5
SC 3.0.0.2	High	Type0	Over	0.5
SC 3.0.1.0	High	Type1	None	0.5
SC 3.0.1.1	High	Type1	High	0.5
SC 3.0.1.2	High	Type1	Over	0.5
SC 3.0.2.0	High	Type2	None	0.5
SC 3.0.2.1	High	Type2	High	0.5
SC 3.0.2.2	High	Type2	Over	0.5
SC 3.0.3.0	High	Type3	None	0.5
SC 3.0.3.1	High	Type3	High	0.5
SC 3.0.3.2	High	Type3	Over	0.5
SC 3.0.4.0	High	Type4	None	0.5
SC 3.0.4.1	High	Type4	High	0.5
SC 3.0.4.2	High	Type4	Over	0.5

SC 3.1.0.0	Low	Type0	None	0.5
SC 3.1.0.1	Low	Type0	High	0.5
SC 3.1.0.2	Low	Type0	Over	0.5
SC 3.1.1.0	Low	Type1	None	0.5
SC 3.1.1.1	Low	Type1	High	0.5
SC 3.1.1.2	Low	Type1	Over	0.5
SC 3.1.2.0	Low	Type2	None	0.5
SC 3.1.2.1	Low	Type2	High	0.5
SC 3.1.2.2	Low	Type2	Over	0.5
SC 3.1.3.0	Low	Type3	None	0.5
SC 3.1.3.1	Low	Type3	High	0.5
SC 3.1.3.2	Low	Type3	Over	0.5
SC 3.1.4.0	Low	Type4	None	0.5
SC 3.1.4.1	Low	Type4	High	0.5
SC 3.1.4.2	Low	Type4	Over	0.5
SC 3.2.0.0	Medium	Type0	None	0.5
SC 3.2.0.1	Medium	Type0	High	0.5
SC 3.2.0.2	Medium	Type0	Over	0.5
SC 3.2.1.0	Medium	Type1	None	0.5
SC 3.2.1.1	Medium	Type1	High	0.5
SC 3.2.1.2	Medium	Type1	Over	0.5
SC 3.2.2.0	Medium	Type2	None	0.5
SC 3.2.2.1	Medium	Type2	High	0.5
SC 3.2.2.2	Medium	Type2	Over	0.5
SC 3.2.3.0	Medium	Type3	None	0.5
SC 3.2.3.1	Medium	Type3	High	0.5
SC 3.2.3.2	Medium	Type3	Over	0.5
SC 3.2.4.0	Medium	Type4	None	0.5
SC 3.2.4.1	Medium	Type4	High	0.5
SC 3.2.4.2	Medium	Type4	Over	0.5

6.2.2.1 Energy Price Profiles

A central point of the scenario setups that stays the same through all scenarios and variations is the selected price profile types. Price profiles can vary greatly and will be subject to future change. We attempt to make the impact of price profiles on the performance of our algorithms assessable by first introducing price-profile-KPIs that can be used to relate any given price profile to our results and then select specific price profiles that represent especially interesting cases for the individual KPIs. This way, it's possible to estimate how different, real price profiles will (most likely) impact the algorithm's performance.
First, we define the following KPIs:

- Mean (μ)
- Standard Deviation (σ)
- Range (the absolute difference between minimum and maximum value)

- Mean of first derivate (μ-diff)
- Standard Deviation of first derivate(σ-diff)
- Spearman's rank correlation coefficient

The first three KPIs are simple, typical values to describe curves. The fourth and fifth KPI (μ-diff and σ-diff) are included to cover the dynamics of the curve, i. e. how sharply the curve, typically, changes. The last KPI, the Spearman rank correlation coefficient, measures the strength and direction of association between two ranked variables, in our case between the rank-based and time-based ranking of time slots in the price curve and is defined as:

Equation 6: Spearmans rank correlation coefficient

$$\rho = \frac{cov(R_p, R_t)}{\sigma_{R_p}\sigma_{R_t}} = 1 - \frac{6\sum(R_p - R_t)^2}{n(n^2-1)}$$

The data source for our price profiles are the freely available sport market prices for the Austrian energy market. We extracted 24720 hourly prices from web sources for the period 2019-01-01 to 2021-10-25. Within this data source, we used scripts to inspect the price curves in chunks that reflect the intended observation period for the experiments. This led to two sets of price curves: one for experiments that only cover a single day and another set of price curves for experiments that cover one work week (Monday to Friday). Please note that our results are produced using single day experiments, as these are more focused and give more insight into the algorithm's performance. All analysis is oriented on a 24-hour resolution, meaning that for the weekly analysis, the 5-day chunks are generated by a moving window that always jumps 24 hours ahead with each step, e. g. the first chunk analyzed is from 2019-01-01 00:00:00 to 2019-01-06 00:00:00, then next chunk is from 2019-01-02 00:00:00 to 2019-01-07 00:00:00 and so on. Next, we performed a manual inspection of the KPIs for each chunk and selected representative dates and weeks that did not collide with obvious outlier dates, like 31st of December. Based on this methodology we extracted four real energy price profiles. We also used a second script to generate an additional, synthetic price profile that contains the median value over the whole observation period for each hourly timeslot, e. g. the median price for 00:00:00 to 01:00:00 over the time years 2019, 2020 and 2021. The following table shows the resulting 5 price profiles we used in our experiments, with their KPIs:

Table 4: Energy price profile types used in experiments and their KPIs

	μ	σ	range	μ-diff	σ-diff	Spearman
Type 0	28.88	1.77	4.95	0.02	1.06	-0.07
Type 1	50.65	5.76	23.07	0.31	5.56	0.21
Type 2	223.02	34.53	142.07	6.41	27.22	0.53
Type 3	28.73	1.34	3.99	0.07	0.75	0.86
Type 4	44.29	4.39	13.35	0.24	3.3	0.16

6.2.2.2 Factory Setups

The factory setup is one parameter in Scenarios 2 and 3 – note: Scenario one uses fully abstract single-step, single-machine products that are designed to simplify the scheduling process. The setup and energy consumption profiles used in Scenarios 2 and 3 are based on the AVL Battery Innovation Center (BIC) in Graz. We adhere to actual production steps and utilize real measurements from production robots for energy consumption data. Machine configurations and task durations are deliberately chosen to optimize

simulation efficiency while maintaining reasonable similarity to the real factory parameters. The production process involves stacking, welding, gluing, and packing of batteries of various capacities, culminating in an end-of-line test, with transfer between stations modeled using Automatic Guided Vehicles (AGVs).

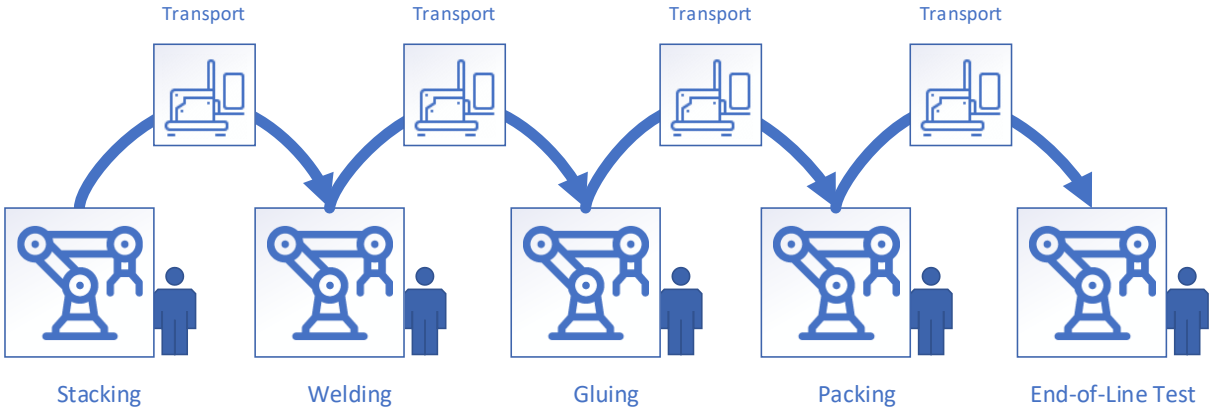


Figure 20: Machine setup for Scenarios 2 and 3

Our algorithm operates under the assumption of readily available raw materials and operators, whose scheduling do not influence algorithm performance. Energy consumption data, including idle consumption, is derived from actual measurements, with Figure 21 illustrating energy consumption during machine operation for the 6 types of operations we model.

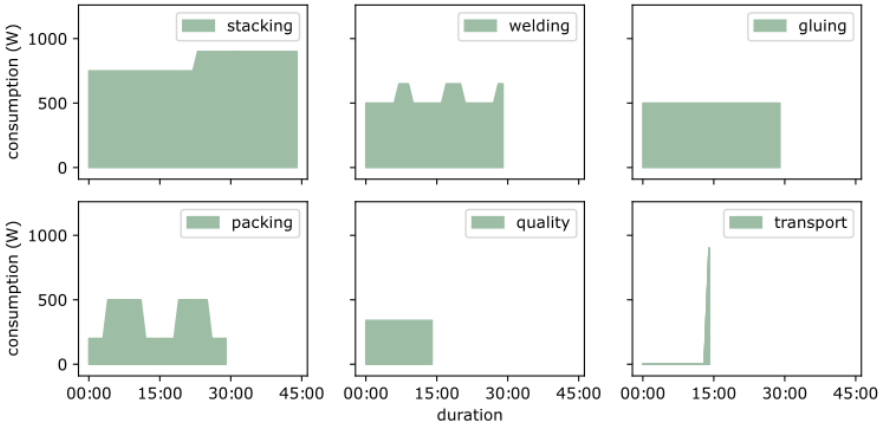


Figure 21: Energy consumption profiles for a battery production in Scenarios 2 and 3

Experimental setup variations include varying numbers of stacking, welding, gluing, and packing robots, end-of-line testing stations, and AGVs, with each machine (except AGVs) operated by a single operator. The production process itself is a single sequence of operations that can only be performed in one specific constellation, illustrated by the dependency graph shown in Figure 22.

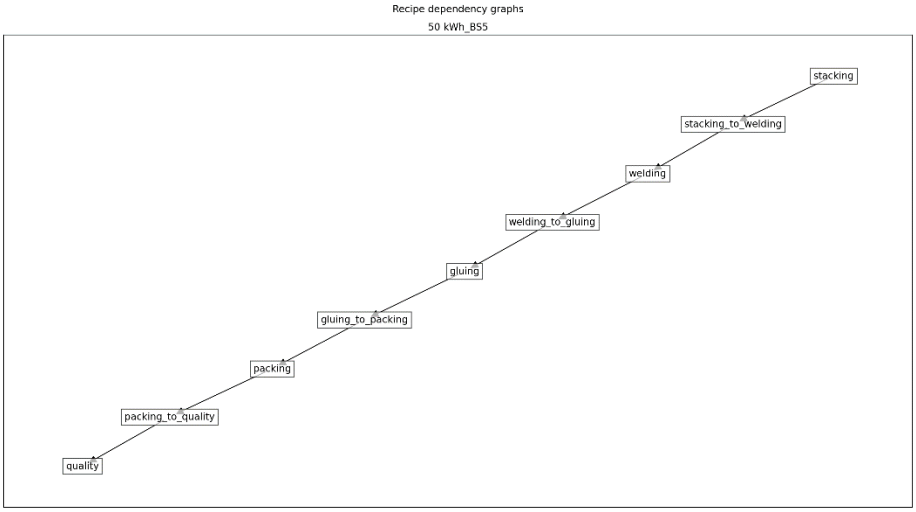


Figure 22: Dependency graph for battery production steps

Furthermore, the use case specifies one specific machine, in our case one specific production robot, for each actual production step and a common pool of AVGs for transport between production stations. The assignment of production steps to machines and operators is shown as relations in the chord diagram in Figure 23.

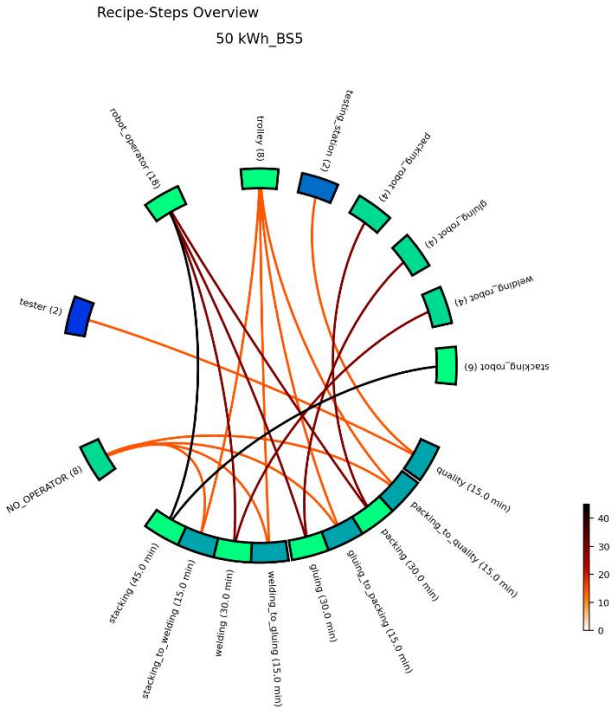


Figure 23: Chord diagram of tasks, machines and operators

Production orders, specifying product type, configuration, quantity, and deadlines are not varying within scenarios and are designed to simulate the production of 98 batteries over two shifts in a single day. The start of the first shift is at 6:00 and the end of the final shift is at 22:00. The shift change is not considered in the simulation at all.

6.2.3 Findings

To quickly recap our goals and our evaluation metrics: the focus of our project is the optimization of production steps towards the increased use of green energy. Due to the effect of renewable energy sources on the energy market (renewable energy generation comes first in the merit order) we assume a fixed relation between the energy price and the availability of green energy, i. e. lower price means more green energy in the energy mix. Based on this we now reformulate the goal of our investigation how and to what extent, we can reduce the energy cost of production, under fixed production parameters like number of products, deadline, total energy consumption, resource cost etc.

The solution we want to investigate in regard to these goals is an algorithmic optimization process that shifts energy intensive production steps from regions with high energy prices to regions with lower energy prices. We assume, even without looking at the immediate results, that the effectiveness of this process, in terms of monetary improvement, will strongly depend on the characteristics of the price profiles in each observation period. Due to this, we decided to generate our evaluation criteria for two different target audiences. Both evaluations will be using results from the same simulation experiments, the difference only lies in the parameterization of the evaluation metric. First, we investigate the algorithmical performance in terms of “shifting potential”. We deliberately want to remove the impact that the spread of the energy price profile will have on the final result. To this end, we calculate the improvement provided within different contexts by directly looking at the amount of energy shifted – as we would in any case – but now weighted only by the difference in cost-sorted rank between source and target slot. The cost sorted rank simply means to sort the slots by energy prices over the day in ascending order and then assigned the slots the numbers from 1, 2, 3, up to 24 in order of that ranking. We title this context of evaluation the Algorithmic Performance. The other evaluation context we look at is the market context. In this evaluation context we look at the final energy production cost of an unoptimized schedule and a schedule optimized by our system and compare the percentage change of energy cost. We refer to this as the Monetary Performance.

6.2.3.1 Algorithmic Performance

The algorithmic performance is evaluated as absolute and relative change in the rank-weighted energy consumption. Below, in Table 5, we summarize the performance of our algorithm, indexed by profile type (rows) and the utilization rate (columns).

Table 5: Algorithmic performance by profile type and utilization in relative and absolute values

profile type	utilization 38.28%		utilization 58.91%		utilization 76.56%	
	rel.	abs.	rel.	abs.	rel.	abs.
type 0	14.78%	-3.00E+07	6.46%	-8.00E+06	0.00%	-4.00E+02
type 1	4.78%	-1.00E+07	0.49%	-9.00E+05	0.01%	-1.00E+04
type 2	2.73%	-7.00E+06	0.32%	-6.00E+05	0.00%	-6.00E+03
type 3	0.00%	0.0	0.03%	-4.00E+04	0.00%	-5.00E+03
type 4	6.03%	-2.00E+07	1.18%	-2.00E+06	0.01%	-1.00E+04

The results table immediately shows the strong impact of the utilization rate. As expected, the lower utilization yields much better performance as it allows more jobs to be moved to cheaper timeslots. Higher utilization comes with less flexibility and drastically limits the optimization potential. There is also a strong dependence on the profile type. The relative performance of different profile types is maintained within a utilization range, with type 0 profile enabling the best performance and type 3 generally bearing the worst

performance. Within high utilization scenarios, the impact of the energy profile type is so diminished, that it is not really feasible to rank the profile types by performance anymore as they all provide very little optimization. When looking at the characteristics of profile types, we notice a potential correlation between Spearman's rank correlation coefficient and performance. It seems that lower Spearman's ρ correlates with higher performance in our experiments. A visual representation of Table 5 is shown in Figure 24.

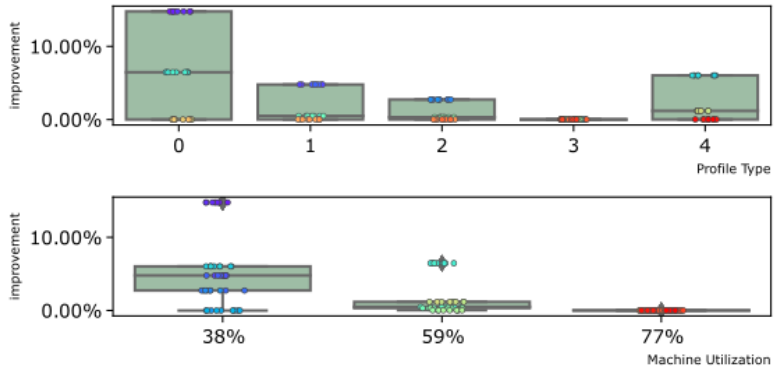


Figure 24: Comparison graph for relative algorithmic performance improvements

Figure 24 shows the relative improvement, split by profile type (upper graph) and machine utilization (lower graph). The green areas in the graph are box plots with whiskers, where the box represents the range from the first to third quantile of the results, with the single line within the box representing the median value. The whiskers (the horizontal lines connected to the box) represent the datapoint farthest from the box boundary that is still within 1.5 times the interquartile range. In addition to the boxes, the plot shows the exact position of each data point via dots – please note that the dot color encodes the iteration number within the experiment, was intended for debug purposes and therefore can be ignored here.

6.2.3.2 Monetary Performance

In contrast to the algorithmic performance, the monetary performance now alters the evaluation to be weighted by actual difference in energy prices. This allows us to estimate the potential monetary saving provided by our algorithm. Table 6 shows a compact summary of experiment outputs for all scenario 2 variants. The values in the table represent the relative change in total energy cost during production. The values are indexed by the price of pre-paid energy (rows) and coverage by pre-paid energy (columns).

Table 6: Relative monetary performance increase by contracted energy prices and amounts

Relative price	prepaid 0%		prepaid 65.63%		prepaid 131.25%	
	μ	σ	μ	σ	μ	σ
75%	0.35%	0.48%	0.43%	0.59%	0.55%	0.75%
100%	0.35%	0.48%	0.35%	0.48%	0.35%	0.48%
125%	0.35%	0.48%	0.30%	0.41%	0.26%	0.36%

When referring to pre-paid energy here we mean a situation where a customer has bought energy-futures ahead of time, which normally provide a certain amount of energy for a fixed price. This is also what the relative price refers to, it is in relation to the average spot market energy price over the observation period. Similarly, the pre-paid amount refers to the amount of energy pre-purchased (and prepaid at a fixed price) energy in relation to the total energy required to fulfill the production order (remember that one of our premises is that we assume that the production process always uses the same amount of energy, we only

shift the timeslots during which this energy is used. For example, consider investigating a single day of production that will consume 100 kW in total and with an average spot market price (over the whole 24 hours) of 10 cent/kWh – then the intersection of 75% relative price and 65.63% prepaid, would refer to experiments where the factory has bought energy futures for a total amount of 65630 Watt (100 kW * 0.6563) at a fixed price of 7.5 cent/kWh (10 cent/kWh * 0.75). An immediate impression of this table is the high σ values, in relation to the corresponding μ values. There is also little variation in the mean values for the two lower prepaid categories (0 and 65%). These two aspects together imply that the observation criteria we use are not well suited to distinguish good from bad experiments. The overall low values (note: the values in the table are all significantly lower than 1% - this is not a typo) already hint to the limited impact our algorithm can have on the actual energy cost during production, but the (in relation) high sigma signifies that the prepaid energy price and amount have little impact on that. Intuitively this can be read as: each intersection of prepaid-price and prepaid-amount contains some of the very high and some of the very low values – giving very uniform means with high standard deviation.

Table 7 shows another compact summary of our experiment outputs for all scenario 2 variants. The values in this table represent the change in total energy cost, indexed by profile type and utilization and are given as the mean relative improvement of all scenario variations that share the specific profile type and utilization and the variance of these values.

Table 7: Relative monetary performance increase by profile type and utilization

profile type	utilization 38.28%		utilization 58.91%		utilization 76.56%	
	μ	σ	μ	σ	μ	σ
type 0	1.05%	0.20%	0.37%	0.07%	0.00%	0.00%
type 1	1.26%	0.29%	0.16%	0.04%	0.01%	0.00%
type 2	0.81%	0.20%	0.11%	0.03%	0.01%	0.00%
type 3	0.01%	0.00%	0.03%	0.01%	0.01%	0.00%
type 4	1.41%	0.31%	0.30%	0.06%	0.01%	0.00%

Table 7 gives more insight into impact factors than the previous Table 6. Grouped by profile type and utilization, we now see significant difference in the performance of different profile types. However, the strong impact of the utilization rate, which was already present our results for the algorithmic performance (see Table 5), is clearly visible again. One significant difference to the algorithmic performance however is that the profile types perform more similarly. This is probably due to the fact that the monetary performance is strongly impacted by the actual energy prices and since we tried to concentrate on typical days with different price profile shapes, the prices during those days are more similar between profiles than the profile shapes. The type 0 price profile is still a competitive contestant, especially since it maintains some optimization potential during medium utilization (~59%), but in low utilization ranges, type 4 and type 1 show more potential. Upon investigation the weaker performance of price curve type 0 is most likely linked to the change between price slots, indicated by the low values for μ, σ, μ-diff and σ-diff. The strong performance of type 4 and type 1 are less clear and would require additional investigation, it is of note however, that profile type 4 is a purely synthetic energy profile that is the median of all observations, indexed by hour-of-day, and could therefore be considered special, in that it is not a profile that actually appears in reality. The least performing price profile is again type 3. Of special note is the price profile type 2, which was chosen for it's high range, but still it allows only low performance improvements. We consider

this a nice example to show that being able to shift between time slots with high price variation is not enough, the proper distribution of these price differences between slots is essential to improve performance.

To make the ranking and relation between experiments clearer we turned the values from Table 7 into box plots in Figure 25.

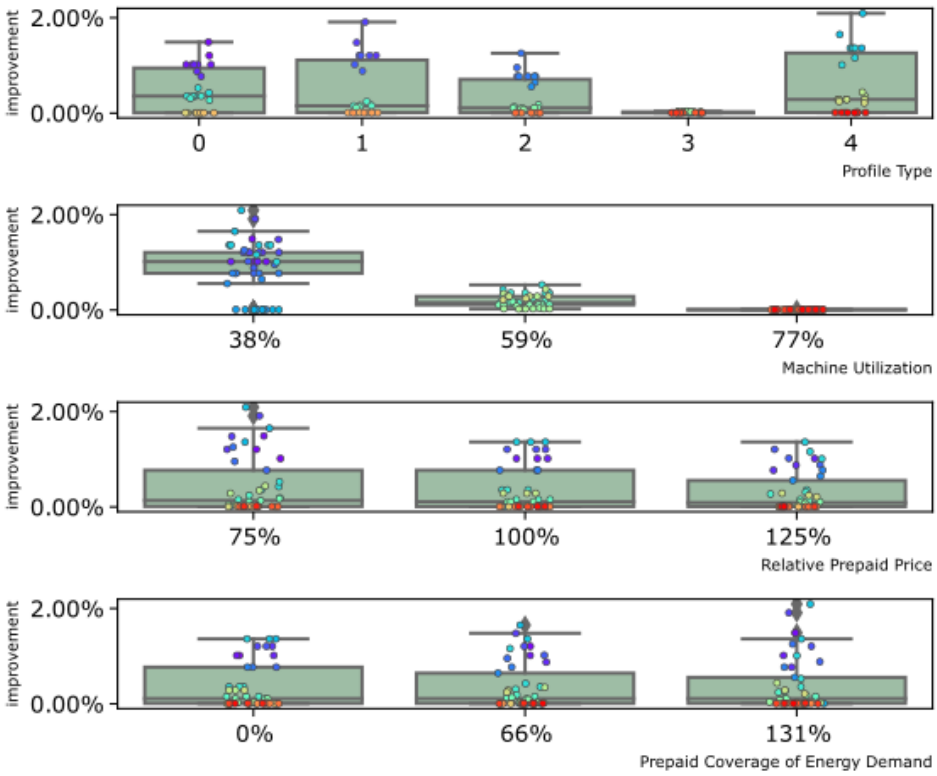


Figure 25: Comparison graph for relative monetary performance improvements

The graphs in Figure 25 use the same visualization format as previous Figure 24, which is typical boxplots overlayed by the individual data points from the experiments. For the monetary performance we also varied the relative price and amount of pre-ordered, fixed price, energy futures. But as was already visible in Table 6, these characteristics do not appear to impact the performance of our algorithm much. The other visualization rows illustrate the results from our previous discussion visually.

6.2.4 Exemplary Step-by-Step Analysis of Exemplary Optimization

For a more detailed examination we now look at the select steps within the optimization process for scenario 2.2.0.0 (medium utilization with profile type 0). The images are split into 4 regions.

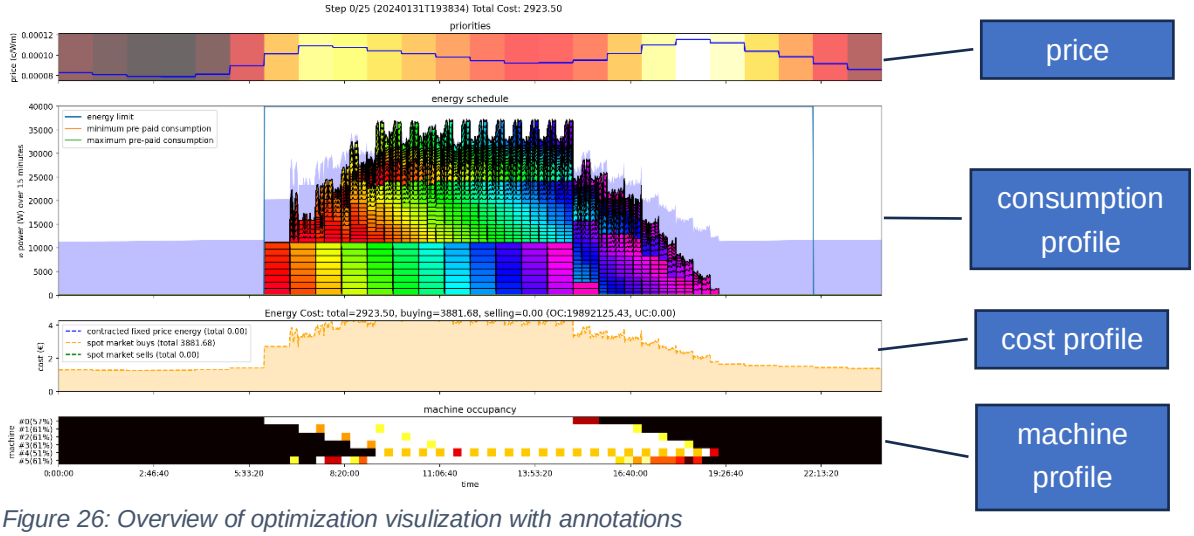


Figure 26: Overview of optimization visulization with annotations

The top region in Figure 26 shows the price profile as a line graph with color coded background marking from the cheapest price (dark color down to black) to the highest price (bright color up to white). The next section is the final (true) consumption profile of the production line – at the current state of optimization. All tasks belonging to a specific product have the same color. Please note that due to the limited amount of available colors, the color coding might repeat. The graph also contains a pale blue area going through the whole observation period. This is the sum of all machine-idle-consumptions – please note that, for simplicity, we assume that machines can go to idle mode, but will never be fully turned off. The third section is the cost profile, here the cost structure of the observation is visualized. The graph distinguished between the cost for energy bought on the spot market (orange area), potential profit made by selling excess energy, either from PV, battery or underused fixed-energy contracts, on the spot market (green area) and finally it also shows the fixed cost from prepaid energy contracts (blue area), like the fixed-energy contracts mentioned above. The lowest section of the graph shows details on the factory utilization. We use the same color-encoding scheme as with the price profile: dark colors mean a low utilization value, bright colors mean high utilization. This kind of visualization is called a heat map. In this specific heat map each row (y-axis) represents a different type of machine. The columns (x-axis) are shared with the other graphs and represent timeslots of the observation period. So, for example, the black regions in the heat map represent periods during which the machines are completely unused, as would be the case outside of production time, for instance. The bright white blocks represent timeslots during which all available instances of a machine type are in use at the same time.

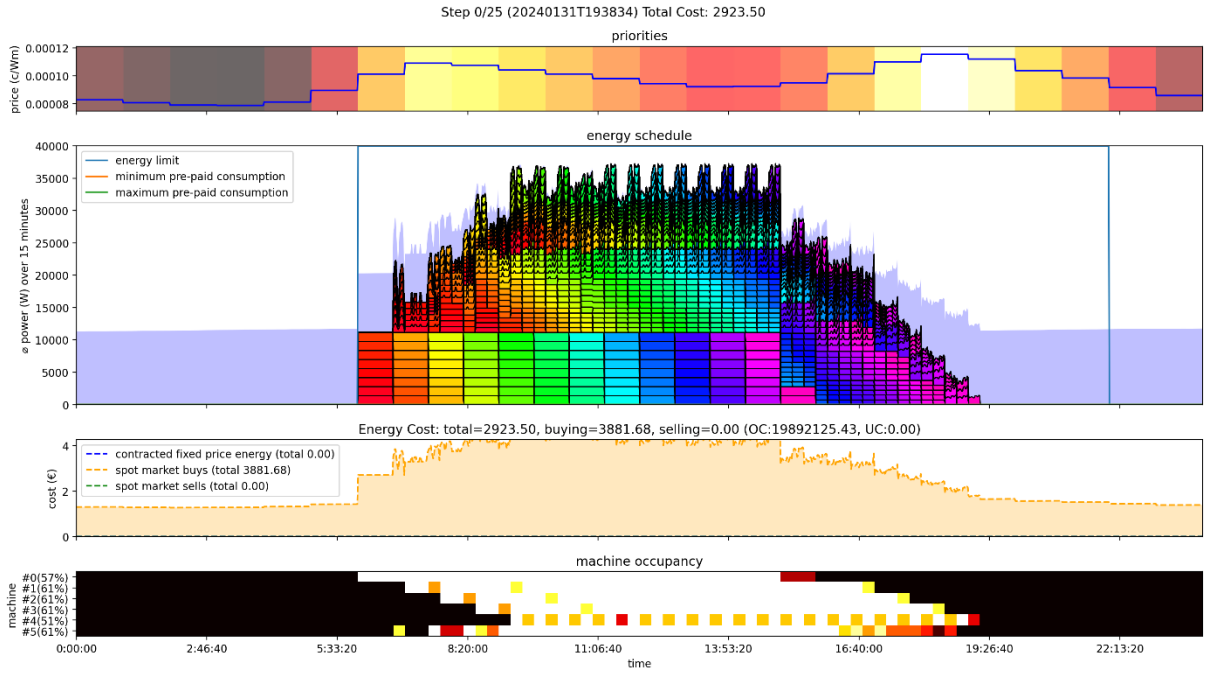


Figure 27: SC 2.2.0.0 detailed analysis - initial schedule

The initial, unoptimized state of SC 2.2.0.0 shows the typical characteristics of a left scheduler. The tasks show a largely uniform energy profile shape, with some bigger consumption blocks that require exclusively machine type 0. Due to the shape of the price curve, the schedule does place several production steps into the high price peaks around 8:00 and 18:00, even though there would still be enough other available machine time.

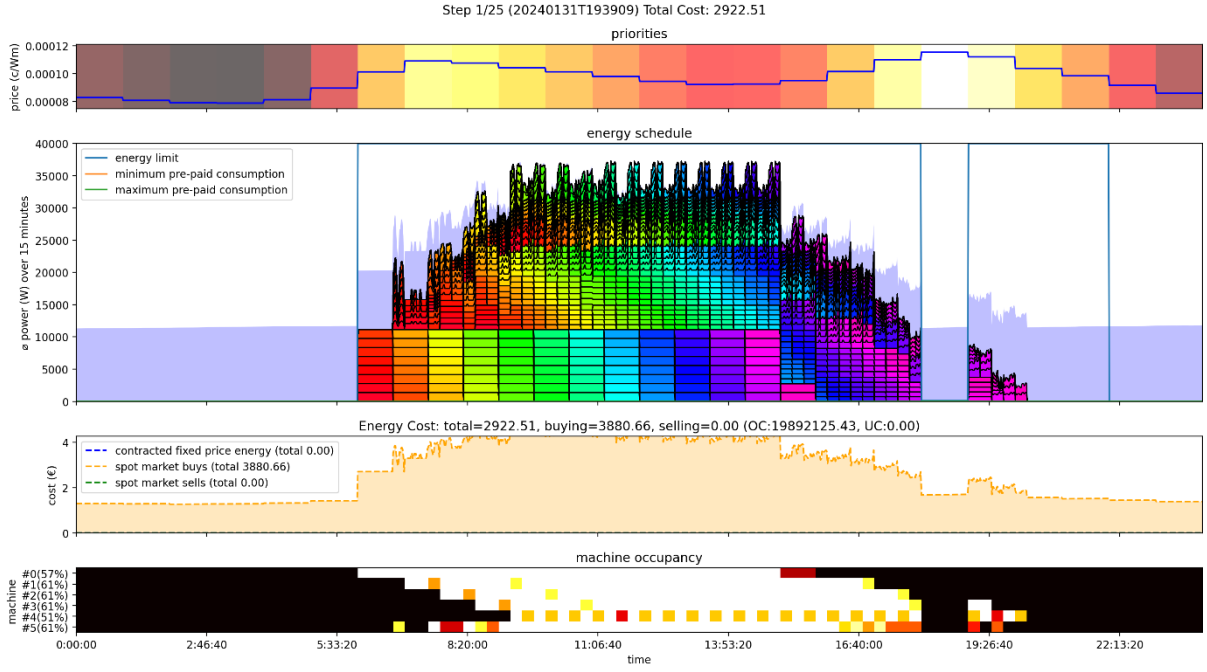


Figure 28: SC 2.2.0.0 detailed analysis - early steps

In a first step, the system restricts access to the most expensive time slot, forcing the MES system to reschedule some tasks to later times.

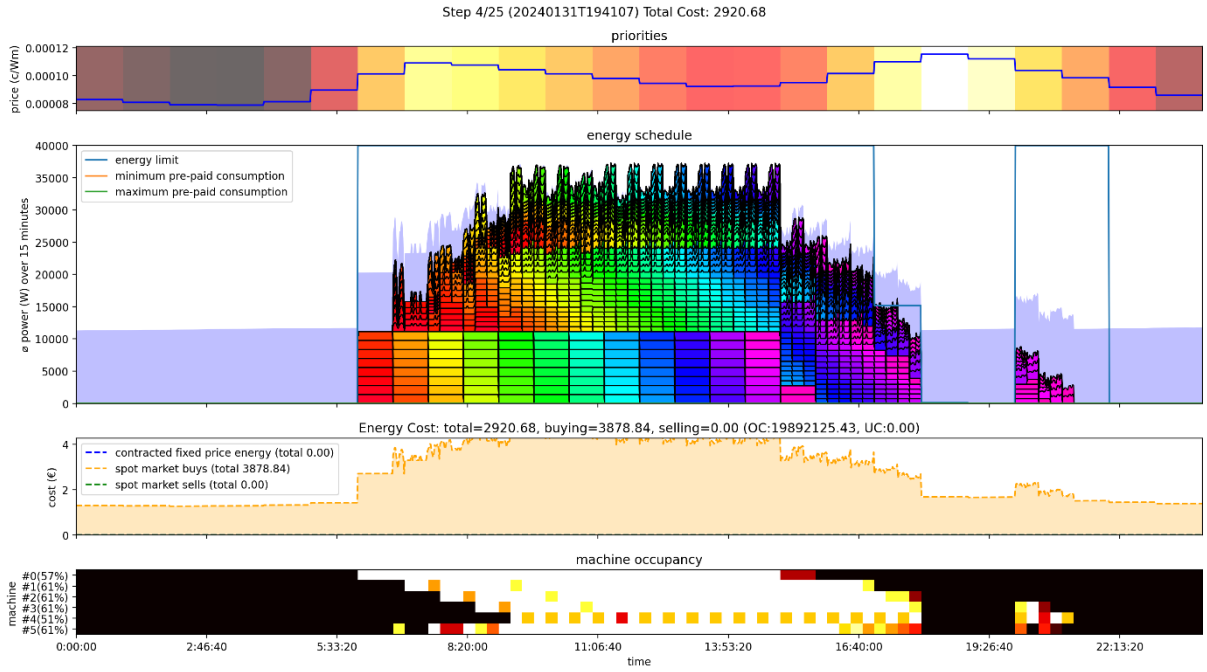


Figure 29: SC 2.2.0.0 detailed analysis - late steps

The next two steps push the production tasks further back and also start to restrict a time slot that can't be fully emptied anymore. The image shows how the consumption limit for the first highest timeslot is pushed approximately around 40% of the available maximum energy. Note that on the left of the schedule there are still empty slots within the production time. The fact that the system already restricts timeslots without being able to fully empty them is due to the fact that our experiment setup has interdependent tasks, i. e. tasks that depend on other tasks and might themselves be the base for further productions steps. Such constellations mean that no MES will be able to fully utilize all available machine time, unless you switch to a fully continued production, which is not scope of this work.

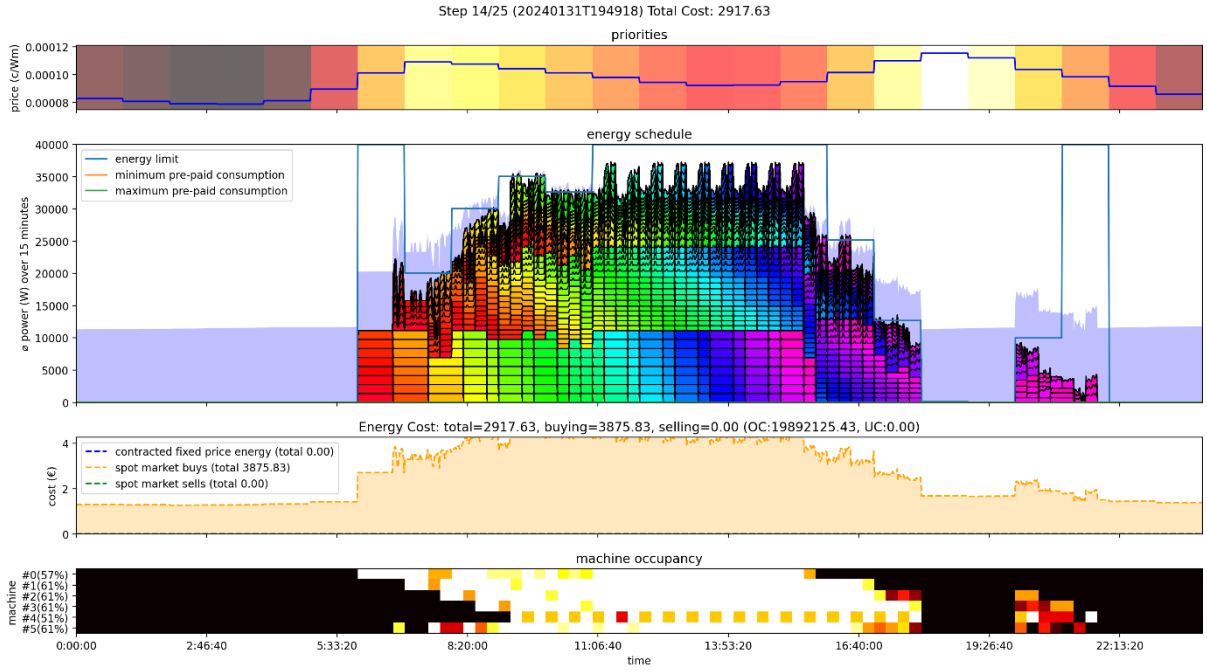


Figure 30: SC 2.2.0.0 detailed analysis - optimized schedule

The final schedule, now as optimized as our system is able to, is similar to the previous one as the limited machine availability becomes the main restricting factor now. The process did, however, successfully reduce the energy cost from 2923.5 to 2917.63. The savings are small but could potentially be much larger if we had a more varied price profile.

6.2.5 Conclusions on Price Aware Optimization Experiments

Based on the findings detailed in subsection 6.2.3, we draw the conclusion that our algorithmic approach has the potential to enhance energy efficiency in production settings under specific circumstances. Optimal monetary savings were observed under conditions characterized by low utilization rates and pronounced variations in timeslot pricing. Notably, when utilization rates were at 38% and the pricing structure favored our approach, a cost improvement of 1.41% was achieved. Conversely, at high utilization rates of 77%, only negligible savings around 0.01% were realized. This pronounced disparity underscores the significant influence of utilization rates on the effectiveness of our approach. Our reliance on the existing scheduling framework within industrial environments limits our ability to finely adjust scheduling parameters. Consequently, we are unable to optimize scheduling decisions, such as swapping production steps with similar durations but varying energy consumption profiles. Thus, the efficacy of our approach is heavily contingent upon the inherent flexibilities present within the existing system.

We introduced two evaluation metrics: one focused solely on the relocation of energy consumption to periods with lower prices, and the other, a monetary metric that factors in the price differentials associated with these shifts. Our findings indicate that while there is a notable capacity to shift energy consumption to periods with lower price brackets, the resultant actual monetary savings are considerably modest. Specifically, instances where our metric for rank-weighted consumption shifts exhibited a 15% enhancement, the corresponding monetary savings remained capped at 1%. Subsequent investigations, extending beyond the purview of this paper, uncovered that this limited realization of monetary savings can be partially attributed to the substantial baseline consumption (ranging between 50% and 70% of total production consumption) inherent in the production process of the battery prototype.

Additionally, our findings underscore the significance of Spearman's ρ as a pertinent indicator for gauging the potential savings associated with price patterns. Notably, we observed a distinct correlation between a high, positive Spearman's ρ and the forward-scheduler system utilized in our study. This correlation suggests compatibility with forward scheduling methodologies, which inherently favor monotonic increasing functions.

In summary, we advocate for the integration of energy price-driven schedule optimization as a fundamental element within Manufacturing Execution Systems (MES) or Enterprise Resource Planning (ERP) systems. Our experimental findings indicate that our approach is particularly well-suited for industrial environments characterized by periods of low utilization rates (below 60% in our experiments) and sufficient flexibility to adapt production steps. Additionally, machines involved in the process should ideally exhibit low idle consumption (averaging 58% in our experimental setup). We observed that intra-day fluctuations in energy prices exert a significant influence on the optimization potential, rendering the approach less advantageous for factories operating under fixed energy pricing structures regardless of the time of day.

Under these specified conditions, it becomes evident that there exists substantial potential for significant cost savings, with even a basic algorithmic optimization demonstrating noticeable efficiency gains. By leveraging Virtual Energy Meters (VEMs) to guide existing schedulers towards energy-price-optimized

schedules, our methodology aligns with setups adhering to the ISA95 standard. While the nature of a factory's production processes may influence the extent of optimization achievable, our approach is broadly applicable across various production modalities, encompassing both continuous and batch production systems.

Our forthcoming endeavors will focus on refining the optimization algorithm and improving computational efficiency. Additionally, we aim to explore novel avenues by integrating battery storage and handling local photovoltaic (PV) production within our algorithmic framework.

7 Outlook

The lower the factory utilization, the bigger the savings potential. In 2021 China put restrictions on the electricity consumptions of factories due to the heat wave. It prioritized the residential usage over industrial. Although the government-controlled economy of China is not the perfect example, during the energy transition phase, it is envisioned that certain restrictions can be put on the industrial energy usage. Since even China, which puts a high emphasis on the industrial production, places priority on residential usage, it's expected that in EU counties, the residential usage would take a priority in case of shortage of electricity.
The rough estimation of the profit margins in the manufacturing sector is under 10% and with energy contributing around 5% of costs the rise in the energy cost can make industries unprofitable and force the temporary shutdown of production.
Higher price fluctuations enable higher benefits of the proposed method. Thae electricity price fluctuations are expected to increase over time due to the growing share of renewables in the energy basket. The years 2021 and 2022 already showed an example of extreme volatility in prices. Although the prices in 2023 calmed down, the average daily fluctuations in Austria for example were nearly 90 EUR/MWh. Considering that out of 54.3TWh of total electricity generation in Austria, wind and solar (the most volatile sources) contributed only 10.6TWh, and that their share in the energy basket will only grow, it is assumed that the daily price fluctuations will only increase.

From a technical perspective, Factories4Renewables provides several points of departure for future research. Our core optimization algorithm assumes stability and some simplicity on the energy provider's side. Within the project scope we developed conceptual adaptations of the optimization algorithm to include grid-fee considerations and peak shaving as goals. Improving these conceptual changes, especially in terms of performance, and finally integrating them with the existing algorithm and implementing a new iteration of the Confidence Estimator would be nature next steps when picking up where this project left of.
Another aspect in relation to this would be to take the optimization algorithm towards higher Technology Readiness Levels, specifically by reducing the number of iterations required for finding the optimum. Our current implementation follows, conceptually, a raster search pattern, which is known for its poor scalability and various projects from different domains have already shown how statistical methods and modern learning techniques can be utilized to replace raster search approaches with more efficient methods.
Another evident point-of-departure for continuing development on the optimization algorithm is to make the algorithm capable of interacting with more complex energy contracts. The basic optimization algorithm provided by the Confidence Estimator should already integrate with higher complexity contracts nicely, as long as they provide at least some potential for optimization. However, our system requires the ability to predict the cost and final energy price for each time slot accurately to work and we have not implemented interfaces that would allow for more complex structures. A similar point of departure is the optimization of multiple production lines. Our work explored the idea of having multiple factories, with varied energy demands, combined to improve flexibility, but we did not provide any definitive results that combine our current, production line specific, optimization with any of the multi-line optimization concepts.

Going beyond extensions to the Confidence Estimator where our research left off, there are points of departure for different goal and use case variations. Specifically, the restriction to profile shapes, which are explored in the project, but as they did not fit the realistic use cases we envisioned. If follow up project can define good scenarios for following consumption profiles, for example to provide grid stability or in special island scenarios, we believe, based on cursory exploration of the topic, that our optimization algorithm could be adopted for this use case with reasonable effort.

Another promising module which was developed in the course of the Factories4Renewables project is the Energy Model, which can predict the power consumption profile of manufacturing processes. More architectures could be tested, and the tested architectures could be refined to improve the prediction accuracy. Also, possible prediction accuracy improvements by using alternative scalar prediction quantities like maximum power and total energy consumption instead of load profile vectors could be explored. For many applications like peak power reduction or energy consumption analysis, scalar values might actually be sufficient and the proposed model's main advantage is the adaptability to many use cases.

While the original purpose of the Energy Model was to determine the energy consumption in certain time slots to shift energy-intensive processes to cheaper time slots, there are numerous possible applications beyond manufacturing schedule adaptation to electricity markets. The model could help identify opportunities to optimize energy use within the manufacturing process by analyzing the predicted consumption profile, and pinpointing areas of high energy demand. The consumed energy could be converted to CO2 emissions and the energy footprint of a product could be assessed. This information can be valuable for manufacturers seeking to improve the environmental sustainability of their products and meet evolving consumer demands for eco-friendly goods. Also, the influence of different production parameters on the energy consumption could be analyzed and more energy-efficient parameter combinations could be explored. Conclusions could be drawn regarding energy-efficient scheduling and reducing CO2 emissions.

8 Contact Information

TU Wien, Institut für Computertechnik (Project Lead)
Project Lead: Thilo Sauter
Address: Gusshausstraße 27-29/E384, 1040 Wien
Telephone: +43 1 58801 38430
E-mail: thilo.sauter@tuwien.ac.at
Website: https://www.ict.tuwien.ac.at/
Project website: https://projekte.ffg.at/projekt/3862076

University for Continuing Education Krems (Academic Partner)
Address: Dr.-Karl-Dorrek-Straße 30, 3500 Krems
Website: https://www.donau-uni.ac.at/

AVL List GmbH (Industrial Partner)
Address: Hans-List-Platz 1, 8020 Graz
Website: https://www.avl.com/

PowerSolution Energieberatungs GmbH (Industrial Partner)
Address: Perfektastraße 77/1, 1230 Wien
Website: https://www.power-solution.eu/