Lars Dannecker, Philipp Rösch, Ulrike Fischer, Gordon Gaumnitz, Wolfgang Lehner, Gregor Hackenbroich

**pEDM: online-forecasting for smart energy analytics**

_Erstveröffentlichung in / First published in:_


DOI: [https://doi.org/10.1145/2505515.2505588](https://doi.org/10.1145/2505515.2505588)

_Diese Version ist verfügbar / This version is available on:_

[https://nbn-resolving.org/urn:nbn:de:bsz:14-qucosa2-806438](https://nbn-resolving.org/urn:nbn:de:bsz:14-qucosa2-806438)
pEDM: Online-Forecasting for Smart Energy Analytics

Lars Dannecker
SAP AG
lars.dannecker@sap.com

Philipp Rösch
SAP AG
philipp.roesch@sap.com

Ulrike Fischer
TU Dresden
Ulrike.fischer@tu-dresden.de

Gordon Gaumnitz
SAP AG
gordon.gaumnitz@sap.com

Wolfgang Lehner
TU Dresden
wolfgang.lehner@tu-dresden.de

Gregor Hackenbroich
SAP AG
gregor.hackenbroich@sap.com

ABSTRACT
Continuous balancing of energy demand and supply is a fundamental prerequisite for the stability of energy grids and requires accurate forecasts of electricity consumption and production at any point in time. Today’s Energy Data Management (EDM) systems already provide accurate predictions, but typically employ a very time-consuming and inflexible forecasting process. However, emerging trends such as intra-day trading and an increasing share of renewable energy sources require a more sophisticated forecasting efficiency. Additionally, the widespread variety of applications in the energy domain poses different requirements with respect to runtime and accuracy and thus, require flexible control of the forecasting process. To solve this issue, we introduce our novel online forecasting process as part of our EDM system called pEDM. The online forecasting process rapidly provides forecasting results and iteratively refines them over time. Thus, we avoid long calculation times and allow applications to adapt the process to their needs. Our evaluation shows that our online forecasting process offers a very efficient and flexible way of providing forecasts to the requesting applications.

Categories and Subject Descriptors

Keywords
Forecasting; Optimization; Online Computation; Maintenance

1. INTRODUCTION
Time series forecasting is an important analysis technique used as a basis for business planning in many application domains [12]. In the energy domain, accurate forecasts are a fundamental prerequisite for balancing energy demand and supply. Forecasting employs mathematical models that use a set of parameters to model the behavior and characteristics of historic time series. To exactly capture the specifics of a time series, the parameters are estimated on a training dataset by minimizing the forecast error using local (e.g., LBFGS) or global (e.g., simulated annealing) optimization algorithms. The estimation is very time-consuming, since it involves a search space that increases exponentially with the number of parameters. In contrast, new energy market dynamics in conjunction with the emerging smart grid technology and an increasing utilization of renewable energy sources require real-time electricity balancing. Research projects such as MIRABEL [18] and McRegio [17] address this issue by proposing novel approaches such as demand response systems, flexible energy requests and real-time energy trading [1]. A fundamental pre-requisite for these approaches is the availability of accurate forecasts at any point in time, which requires a very efficient calculation and provisioning of forecasts.

EDM systems are typically accessed by a wide variety of applications, where some need their results after a certain time frame and others require the best possible accuracy. A utility company responsible for balancing an electricity transmission grid (transmission system operator–TSO) needs to calculate forecasts at any point in time. It requires the results as fast as possible to quickly recognize possible imbalances and critical situations. However, they do not require the best possible accuracy to start first measures such as changing energy flow directions or starting-up backup power plants. Furthermore, real-time trading gains more and more importance and exhibits short lead times for calculating forecasts. The timing is more critical than the accuracy, as quick reactions on changing availability situations are key for successfully closing deals. Since, previously forecasts were mainly calculated one day ahead, the current forecasting process is rather static and was not designed for application-awareness and continuous forecasting.

We tackle the requirements of constantly available forecasts and application-awareness by introducing our novel online forecasting process implemented as part of our EDM system called pEDM. The online forecasting process uses an iterative parameter estimation in conjunction with a special forecast model materialization to rapidly provide accurate forecasts and improve them over time. Applications can subscribe to those improvements and define runtime constraints and accuracy targets to flexibly adapt the process to their needs. While we exemplarily describe our approach for the energy domain, it can also be used in other domains such as sales or traffic management.

©2013 Copyright held by the owner/authors. Publication rights licensed to ACM. This is the author’s version of the work. It is posted here for your personal use. Not for redistribution. The definitive version of Record was published in CIKM ’13, Oct. 27–Nov. 1, 2013, San Francisco, CA, USA. http://dx.doi.org/10.1145/2505515.2505588.
2. CURRENT STATE OF FORECASTING

Currently, time series (TS) forecasting follows a distinct process illustrated in Figure 1 and consists of (1) the forecast calculation and (2) the model maintenance [2]. The calculation starts with identifying the most suitable forecast model (FM). Despite some advisor techniques such as the (partial) auto correlation function (PACF), there is no analytical approach that can ultimately choose the best model solely based on the given data. Thus, typically it is necessary to process a set of candidate models. Afterwards the parameters of the candidate models are estimated, which is typically very time consuming. The last step is the actual calculation of the forecast. Overall, there are three typical strategies. First, the manual strategy where a human expert manually selects the forecast model and the respective parameters. Due to the human involvement this strategy is apparently slow. Second, the heuristic strategy employs heuristic techniques to automatically identify promising forecast models and to estimate their parameters. This strategy starts from scratch for each forecasting calculation and thus, requires a long time to provide results. Third, the last model strategy starts the optimization from the least recently used model instance [6]. This strategy allows a more efficient calculation, but focuses on a single model only and does not cover possible better results from other models. Overall, the conventional calculation strategies are either time-consuming or pose the risk of starving in a local optimum. Additionally, they provide a forecast only after finishing the entire process and thus, applications cannot control the progression.

In the maintenance part the forecast model is adapted to the most recent time series developments. First, we evaluate the forecasting accuracy using an error metric (e.g., SMAPE [15]). Second, we adapt the models, which typically involves one of the calculation strategies. Hence, it is almost as expensive as the initial calculation. There are three common trigger strategies for the adaptation. First, we evaluate the model when receiving a request (on request strategy), which requires the least resources, but causes the longest response times. Second, we adapt the model when adding a time series value (On update strategy). This strategy always provides the most accurate model, but also exhibits high system loads. Third, we adapt the model periodically after a given time period or when violating an error threshold. However, finding suitable thresholds or periods is very complicated. To address this issue time periods and error threshold can be combined [6]. Overall, the conventional maintenance strategies either exhibit long response times or very high system load. Even the more advanced periodically strategy poses the risk to degenerate, when choosing non-beneficial thresholds.

3. CALCULATING FORECASTS ONLINE

With our novel online forecasting process we specifically target to allow a more efficient processing of forecasting requests and likewise support for applications to flexibly adapt the forecasting process to their individual needs. The online forecasting process addresses the challenges described in Section 2 by enhancing and combining the individual forecasting calculation and maintenance strategies. For improving the forecasting calculation, we combine the advantages of the last model strategy with the heuristic strategy. For this purpose, we store previously used model instances (last model strategy) related to different forecast models (heuristic strategy) in a forecast model repository. We define the term forecast model FM as the type of a forecast model (e.g., autoregressive models and exponential smoothing models). A forecast model (FM) with a given parameter combination (P) is called a forecast model instance FM,P. Each time a model adaptation is triggered, we determine from the repository the most accurate instances and provide them to the subsequent optimization. Thus, we consider multiple forecast models and starting points, but limit their number by reusing only the most accurate instances.

For the maintenance part we combine the on-request and the periodically strategy. The on-request strategy provides very accurate forecasts, since it involves the most recent time series values. At the same time, it is also the most time-consuming strategy. However, as part of our online forecasting we iteratively provide improved forecasts and thus, avoid long waiting times until results are delivered. We also avoid the starvation in local minima with respect to forecast models and starting points by adding the periodically strategy. This strategy works as a global coverage that uses global optimization algorithms and considers all forecast models available in the forecasting system. With that we search for suitable parameter combinations in areas of the search space not covered by local algorithms and also avoid missing better solutions provided by forecast models currently not considered in the forecast model repository.

Figure 2 illustrates the online forecasting process in detail. Prior to executing the actual online forecasting process, we have to conduct a one-time initialization (Step 0), to fill the forecast model repository with an initial set of forecast model instances. The initialization is required only once when adding a new time series and is executed using the heuristic strategy (compare Section 2). Afterwards, the process starts from our forecast model repository containing previously used forecast model instances (FM,P). From this repository we select candidate instances with a case-based reasoning approach (Step 1 and Section 5.1) and use the most accurate instance to quickly calculate a first forecast. This forecast is directly provided to the application, which means that it quickly receives a first result (from our experiments in around 1 ms). Subsequently, we iteratively refine the candidate model instances by re-estimating their parameters using our flexible local optimization process (Step 2 and Section 4). More accurate instances found during the optimization are used to calculate improved forecasts that are provided to subscribed applications. Applications can terminate the subscription at any time or define runtime constraints and accuracy targets to influence the progression of the process. Subsequently, we conduct our global coverage strategy (Step 3 and Section 5.2), which serves as the maintenance for our repository.

4. FLEXIBLE LOCAL OPTIMIZATION

The iterative optimization is the core part of our online forecasting, where we estimate the instances provided by the forecast model repository. We start the optimization from multiple starting points (i.e., forecast model instances) to cover multiple areas of the search space and to reduce the risk of getting stuck in a local optima. Furthermore, we use multiple optimization algorithms simultaneously to avoid manually choosing an algorithm that might not find...
the best solution. Since, we allow subscribed applications to define runtime constraints and to terminate the process subscription at any time, we execute the optimization runs (i.e., the combinations of model instances and optimization algorithms) in an order that reflects their potential to find improvements. We determine the potential by ranking the optimization runs using their expected runtime and accuracy, where we favor optimization runs that are expected to deliver the best improvements in the shortest runtime.

### 4.1 Handling Application-Defined Constraints

With the option to define runtime constraints and accuracy targets, we equally support applications that require a final result after a specific time or that target a certain accuracy. Runtime limits are handled as hard constraints and the optimization is canceled with its current result as soon as the runtime limit is reached. In addition, we only execute optimization runs that are expected to finish in the given time limit. Thus, a runtime constraint is especially useful when the receipt of a forecast is time critical, but the best possible accuracy is not required. Since it is not possible to guarantee a certain forecasting accuracy, applications may only define a target for the accuracy. In presence of an accuracy target we finish the optimization as soon as we reached the defined accuracy. This is especially useful for applications, which can accept a certain forecast error. If the target cannot be reached, the optimization finishes after the optimization converged or the subscription is terminated.

### 4.2 Ranking of Optimization Runs

To rapidly provide significant improvements, we determine the most beneficial execution order of the optimization runs by ranking them based on their expected runtime and accuracy. To calculate the ranking we combine the accuracy of the candidate forecast model instance with the expected runtime and accuracy of the assigned optimization algorithm. The ranking is illustrated in Figure 3.

**A: Record model-specific statistics**

The basis for estimating the expected runtime and accuracy of the optimization algorithms are statistics recorded during the initialization our online forecasting process (see Section 3). Since the optimizer runtime and accuracy highly depend on the model and the starting point, we record individual statistics for each combination of forecast models \( FM_i \) and local optimizers \( O_j \). In addition, the initialization involves multiple starting points. The expected accuracy of an optimizer is the median of the forecast errors \( \hat{e}_{opt} \) over all starting points \( P_r \). We chose the median over the average, due to its robustness against outliers. For the expected runtime, we chose the more pessimistic 3rd quartile of all recorded runtimes \( t_r \), since violating runtime constraints is more critical. We update the statistics after executing the online forecasting.

**B: Assign optimizers to instances**

From the forecast model repository we receive the selected instances ordered by their current forecast errors. We now assign the recorded optimizer statistics to the instances with respect to the involved forecast model. Thus, the instances involving for example forecast model \( FM_1 \) (i.e., \( FM_1 P_r \)) consider the statistics of all optimizers that were recorded for this model.

**C: Calculate the execution ranking**

In the third step all three components—the measured accuracy \( e_i \) of the candidate instances as well as the runtime \( t \) and accuracy \( \hat{e}_{opt} \) of the optimization algorithms—are combined to the ranks of the optimization runs. A rank reflects the expected evolution of the forecast error over the runtime of an optimization run. Accordingly we can determine the dependency between the accuracy and the runtime and use it as a calculation rule. The most simple option is to assume a linear dependency, which would result in the calculation rule \( e_i \cdot t \). However, it equally weights runtime and accuracy, despite the fact that the ultimate goal is to provide the best possible accuracy rather than the shortest runtime. Thus, it would be hard for very accurate algorithms to compensate for a potentially longer runtime. We also observed in our experiments that the runtime of an optimization run asymptotically scales as \( t = O(\exp(C/e)) \) with decreasing error \( e \), where \( C \) is a parameter characteristic for the algorithm at hand. Accordingly, we assume \( t = A \exp(C/e) + f(e) \) where \( A \) and \( C \) are parameters, and \( f \) is a correction to the leading exponential behavior that can be neglected in the limit \( e \to 0 \). Figure 4 presents a plot of \( \ln t \) vs. \( 1/e \) for the Nelder Mead algorithm optimizing the Triple Seasonal Exponential Smoothing model [22] on the NationalGrid dataset (compare Section 6). We observe the expected linear behavior for small values of \( e \); the parameters \( C \) and \( A \) are easily extracted from the slope and intercept of the straight line (red color) fitting \( \ln t \) for \( 1/e \to \infty \).

We got similar results for other models, optimization algorithms and datasets. Asymptotically, for small forecast errors, runtime and accuracy thus depend on both parameters \( A \) and \( C \), which are characterized by the forecast model and optimization algorithm of an
optimization. The slope \( C \) describes the pace an algorithm reduces the forecast error. The smaller the slope, the faster converges the optimization. The offset \( A \) describes the minimal runtime to find a first result. For our ranking the pace of an algorithm turns out to be the more important parameter, as the offset changed only slightly across algorithms. Thus, omitting the offset we compute the rank of an optimization run as \( C \leftarrow \varepsilon \cdot \ln(t) \). It is important to note that the expected accuracy of an optimization run is characterized by two forecast errors—the error of the instance \( \varepsilon_i \) and the expected error of the optimization algorithm \( \varepsilon_{\text{opt}} \). To include both errors, we substitute \( \varepsilon \) by the product of both forecast errors \( \varepsilon_i \cdot \varepsilon_{\text{opt}} \). We further replace \( t \) by \( t + 1 \) preventing negative values from the natural logarithm. As a result, we finally arrive at \( C = (\varepsilon_i \cdot \varepsilon_{\text{opt}}) \cdot \ln(t + 1) \) that we use to compute our ranking. The lowest rank represents the optimization run expected to find the largest improvement in the shortest time.

**D: Executing the Optimization Runs**

For eventually executing the optimization runs, we enqueue all optimization runs in the order of the ranking in a task queue. The system then assigns a number of threads that process the queue in parallel by always picking the optimization run currently on the top of the queue. As soon as an optimization run found an improvement the refined instance is transmitted to the system. If the refined instance is an improvement over the currently best instance, the system calculates an improved forecast that is afterwards provided to the application. After all optimization runs converged or the application terminated the subscription, the local optimization calculates the final forecast if necessary and adds best instance found to the forecast model repository. In addition to the forecasts, we always transmit the currently achieved forecast error, which can be used to evaluate the quality of the forecast and to create confidence intervals around the prediction.

5. Process Enhancements

Besides the flexible optimization our online forecasting process executes further components—namely the forecast model repository preserving previously used instances and the global coverage.

5.1 The Forecast Model Repository

Our pEDM system uses a case-based reasoning approach to identify the most suitable forecast model instances for the subsequent optimization [7]. Case-based reasoning means to solve new problems under consideration of solutions for previously encountered similar problems. The forecast model repository serves as our case base containing previously used forecast model instances. To determine the instances that best matches the current time series development, we exploit the time series context, i.e., background processes and influences affecting the time series behavior. Examples are meteorological, calendar (seasonal) and economic influences. Changes in the time series context often cause changing optimal model parameters, but similar contexts in most cases lead to similar optimal parameter settings. Thus, we search our repository for instances that previously produced accurate forecasts in a context similar to the current one. Afterwards, we assess the accuracy of the identified instances and refine only the most accurate ones. In contrast to our previous work [7] we now support multiple forecast models instead of only the parameters of a single model.

5.2 Global Coverage

With the global coverage we aim at reducing the risk of missing better solutions in search space areas not covered by local algorithms and provided by forecast models currently not included in the forecast model repository. For this purpose, we execute multiple global search algorithms such as simulated annealing, with the goal of finding additional promising instances, i.e., instances that provide a better accuracy than the worst instance in the repository. We execute each global optimization algorithm on one randomly assigned forecast model to over time eventually cover all combinations of models and global optimizers. Promising instances found are directly added to the repository and hence, are considered in future executions of the online forecasting process. Thus, the global coverage serves as the maintenance for our forecast model repository. It is executed asynchronously to forecasting requests as a background process and only when free system resources are available. We can pause the global coverage at any time.

6. Experimental Evaluation

In our evaluation we show that our online forecasting efficiently provides accurate results and iteratively improves them over time.

6.1 Experimental Settings

We use the multi-equation EGRV model [21] and Taylor’s triple seasonal exponential smoothing model (ESM) [22]. For the optimization we use the local algorithms LBFGS [4] and Nelder Mead Downhill Simplex [20] as well as the global algorithm simulated annealing [16]. To allow a meaningful evaluation as well as readable results, we limited the number of models and optimization algorithms used in our online forecasting process. However, our pEDM system supports a much broader variety of models and algorithms. The evaluation is based on two real-world datasets: NationalGrid: UK National Demand [19]: Electricity demand. January 1st 2002 to December 31st 2009, 30min resolution. MeRegio: Single Appliance Demand [17]: 87 customers aggregated, November 1st 2009 to June 30th 2010, 60min resolution. As evaluation environment we used an Intel Core i7-870 (2.93 GHz), 16GB RAM, 320GB HDD, Windows 7. Our prototype is written in C++ (GCC 4.6.1) and uses Intel TBB (4.1) for the parallelization (8 threads).

In our experiment we compare our online forecasting against the last model strategy and the estimation from scratch. The last model strategy estimates the last used instance of a model using one local optimizer at a time. The estimation from scratch estimates both
models by successively executing all optimizers, starting with simulated annealing. We empirically determined 60 s (NationalGrid) and 10 s (MeRegio) as the optimal runtimes for the global optimization. It is important to note that neither the last model strategy nor the estimation from scratch is an iterative process. Thus, forecasts are only calculated from the final instance after the entire process finished. However, to still show the error development, we provide the forecast error of the last used instance and the intermediate results. From the NationalGrid dataset we used the years 2002 to 30.04.2007 for the initialization. Afterwards, we recorded the error development at distinct evaluation points. The evaluation points selected for presentation are (1) May 5th 2007, 21:30 and (2) September 1st 2008, 0:30. For the MeRegio dataset we used November 2009 to April 2010 for the initialization. As evaluation points we chose (1) April 13th 2010, 1:00 and (2) May 21st 2010, 16:00h. We observed similar results for all other evaluation points.

### 6.2 Strategy Comparison Results

The results are illustrated in Figure 5. We used a logarithmic scale for the runtime, to compensate for the large runtime differences. On both datasets, our online forecasting showed the best error development. The first model instance was always available after around 1 ms and provided a better or on par accuracy compared to the other strategies. Thus, already our forecast model repository is able to almost instantly provide a first very accurate forecast.

The results for the NationalGrid dataset are illustrated in Figures 5(a) and 5(b). At the first evaluation point (Figure 5(a)) the online forecasting process directly (after 1 ms) selects the most accurate forecast model instance, which is only slightly improved from 0.683 % to 0.677 %. Similarly, the EGRV last model strategy starts from an error of 0.695 % for both optimizers and did not find any improvement. However, the EGRV model required 3,947 ms (Nelder Mead) and 48,518 ms (LBFGS). When estimating the EGRV model from scratch, we reached a comparable accuracy (0.680 %), but required a long runtime to provide a result (EGRV: 791,527 ms). This is a clear disadvantage of this strategy. The ESM model did not reach a comparable accuracy (best: 0.838 %) independent of the used strategy and exhibited a long runtime (108,139 ms) when estimated from scratch. At the second evaluation point (Figure 5(b)), we can observe the advantageous error development of the online forecasting. The process starts from a forecast error of 1.230 %, which is iteratively improved to 0.885 %. None of the competing strategies provides a better accuracy at any point in time. Together with the last model strategy the ESM model provides a lower forecast error (ESM-LBFGS: 0.971 %; EGRV-LBFGS: 1.167 %) in a shorter runtime (ESM-LBFGS: 4,414 ms; EGRV-LBFGS: 48,518 ms) compared to the EGRV model (similar results for Nelder Mead). Considering both evaluation points the results clearly show the advantage of considering multiple forecast models, since the most accurate model might change from time to time. The results for the from scratch estimation are similar to the first evaluation point, but this time the ESM model exhibits a final accuracy similar to the online forecasting process (0.959 %). The time for providing a result is still much higher (Online first: 1 ms; Online last: 51,041 ms; From scratch ESM: 180,726 ms).

For the MeRegio dataset we observed similar results, illustrated in Figures 5(c) and 5(d). At the first evaluation point (Figure 5(c)), our online forecasting process and the EGRV last model strategy (Online: 3.843 %, EGRV-Nelder Mead: 4.07 %) exhibited a similar final accuracy, but the EGRV model needed 424 ms compared to 1 ms required by our online forecasting process. The ESM last model strategies exhibited a worse forecast error (both: 6.2 %). The from scratch strategies reached a similar accuracy in a much longer time. At the second evaluation point (Figure 5(d)) the final results of the ESM last model strategy were close to the online forecasting process (online: 4.175 %, ESM: 4.189 %). The online forecasting process, finds a first result after 1 ms and improves it after only 12 ms (5.354 % → 4.184 %). The final forecast is provided after 1,201 ms (4.184 % → 4.175 %). The ESM last model strategy required 15 ms for the Nelder Mead and 249 ms for the LBFGS to provide a result. Additionally, we observed large accuracy deviations between optimizers. While for the EGRV last model strategy the Nelder Mead algorithm provided an error of 4.329 %, the LBFGS algorithm arrived only at 5.869 %. This clearly motivates to consider multiple combinations of optimization algorithms and model instances as we do in our online forecasting process.
Overall, our online forecasting process provides a very efficient and flexible way to calculate accurate forecasts. The first result is already very accurate and delivered in around one millisecond. In contrast to the other forecasting calculation strategies, the optimization of the online forecasting process involves multiple forecast models in parallel. With that we substantially increase the probability of finding the most accurate forecast model instance.

7. RELATED WORK
The increasing amount of data and the requirement for ad-hoc analytics lead to an increasing interest to integrate statistical methods into databases [5]. Hereby, we observe two main research directions. The first direction reuses existing statistical tools and tries to improve the interaction with the database [8, 13]. These approaches are fast and easy to realize, but do not allow optimizations on the forecasting process. The second direction employs special purpose systems that tightly integrate statistical methods. One example is SciDB [3], which provides an optimized array storage to increases the execution efficiency. However, they do not provide results iteratively, causing longer response times. Forecasting in traditional databases was introduced with the Fa System [9], where the model identification is optimized using an incremental approach to build models for multi-dimensional time series. Still, they do not directly optimize the overall forecasting efficiency.

Later, Ge and Zdonik proposed a skip-list approach [11] to efficiently determine a suitable history length of very large time series. However, the approach is only applicable for regression models, while in the energy domain more complex models arise. Finally, in an earlier work, we introduced F²DB that natively integrates forecasting in PostgreSQL [10]. However, Fischer et al. integrated a conventional forecasting process and do not provide online forecasting capabilities. The iterative improvement of forecasting results was inspired by the online calculation of aggregation queries [14]. Instead of waiting for the final result, they allow users to receive intermediate results together with some confidence interval.

Overall, our online forecasting is the first approach that offers an application-aware, iterative improvement of forecasting results.

8. CONCLUSIONS
In this paper, we proposed our novel online forecasting process that we use as part of our EDM system pEDM. The core of our online forecasting is a case-based reasoning approach in the form of our forecast model repository combined with our flexible, iterative model optimization. Our pEDM system allows applications to influence the progression of the forecasting process by defining runtime constraints and accuracy targets. Thus, we equally serve applications that require forecasts in a short amount of time or with the best possible accuracy. In our evaluation we demonstrated the advantages of our online forecasting process. We were able to always provide a first, already accurate forecast in below 1 millisecond. Further improvements were calculated iteratively and provided the best possible accuracy in all cases. As a result, we provide an efficient forecasting process that enables applications to work in the face of the new energy market requirements.

In the future, we want to increase the calculation efficiency using optimized storage layouts and further address the handling of extreme cases challenging the execution of our online forecasting.

Acknowledgment
The work presented in this paper has been carried out in the MIRABEL project funded by the EU under the grant agreement number 248195.

9. REFERENCES