# Nemesyst: A Hybrid Parallelism Deep Learning-Based Framework Applied for Internet of Things Enabled Food Retailing Refrigeration Systems

George Onoufriou[a], Ronald Bickerton[b], Simon Pearson[c], Georgios Leontidis[a,*]

[a]*School of Computer Science, University of Lincoln, Brayford Pool Campus, LN67TS, Lincoln, UK*
[b]*School of Engineering, University of Lincoln, Brayford Pool Campus, LN67TS, Lincoln, UK*
[c]*Lincoln Institute for Agri-Food Technology, University of Lincoln, Brayford Pool Campus, LN67TS, Lincoln, UK*

## Abstract

Deep Learning has attracted considerable attention across multiple application domains, including computer vision, signal processing and natural language processing. Although quite a few single node deep learning frameworks exist, such as tensorflow, pytorch and keras, we still lack a complete processing structure that can accommodate large scale data processing, version control, and deployment, all while staying agnostic of any specific single node framework. To bridge this gap, this paper proposes a new, higher level framework, i.e. Nemesyst, which uses databases along with model sequentialisation to allow processes to be fed unique and transformed data at the point of need. This facilitates near real-time application and makes models available for further training or use at any node that has access to the database simultaneously. Nemesyst is well suited as an application framework for internet of things aggregrated control systems, deploying deep learning techniques to optimise individual machines in massive networks. To demonstrate this framework, we adopted a case study in a novel domain; deploying deep learning to optimise the high speed control of electrical power consumed by a massive internet of things network of retail refrigeration systems in proportion to load available on the UK National Grid (a demand side response). The case study demonstrated for the first time in such a setting how deep learning models, such as Recurrent Neural Networks (vanilla and Long-Short-Term Memory) and Generative Adversarial Networks paired with Nemesyst, achieve compelling performance, whilst still being malleable to future adjustments as both the data and requirements inevitably change over time.

*Keywords:* Deep Learning, Databases, Distributed Computing, Parallel Computing, Demand Side Response, Refrigeration, Internet of Things

## 1. Introduction

Intelligence derived from data has the potential to transform industrial productivity. Its application requires computational solutions that integrate database technologies, infrastructure and frameworks along with bespoke machine learning techniques to handle, process, analyse, and recommend. Recent advances in deep learning are accelerating application.

Deep learning is a sub-field of machine learning that seeks to learn data representations, usually via a variant of artificial neural networks. Deep learning is concerned with two distinct tasks:

- Training; where the deep neural network (DNN) learns the data representation through iterative methods, such as stochastic gradient descent. This is a very time consuming operation that usually occurs at most on one node, or a few CPU /GPU nodes.

---

*Corresponding author. Tel.:+44(0)7951053731

*Email address:* `gleontidis@lincoln.ac.uk` (Georgios Leontidis)

- Inference; where the learned DNN can be used in one form or another to make predictions, detection, localisation, etc. This is usually faster than training as it does not require as many operations, but still requires a good portion of the same resources and time. [1]

Both training and inference can be improved by the use of 'better' hardware, including faster and/or higher volume processing via distributed/parallel systems, where parallel in this instance means any single system that according to Flyns taxonomy is using either:

- single instruction, multiple data (SIMD)

- multiple instruction, single data (MISD)

- multiple instruction, multiple data (MIMD)

This is separate from a distributed system, where each computing node is a different networked computer, as opposed to a single computer with multiple CPUs/GPUs for computation in parallelisation. Currently the majority of deep learning frameworks have matured in parallelisation, but they lack or are still in their infancy with regard to distribution. Novel solutions such as TensorFlow-distribute, or IBMs distributed deep learning are now emerging to resolve this issue [1]. The ability to share data between distributed nodes using deep learning frameworks is a critical bottleneck, despite the prevalence of databases providing very mature and advanced functionality, such as MongoDB's aggregate pipeline. In addition, the unification of deep learning interfaces to databases will aid reproducibility, as it will standardise the message protocol interface (MPI) and describe what data in what manner is used for training and inference; in MongoDBs case, by using aggregate pipeline json files. The integration of deep learning into database models is crucial for any (near) real-time application, such as Netflix, Amazon, Google, as 'rolling' a bespoke data management system or MPI is costly and can result in inferior performance and capabilities.

In this paper, we propose a novel end-to-end distributed framework that can handle any type of data, scales and distributes models across various nodes locally or on the cloud. We demonstrate the approach by developing a deep learning infrastructure to link the power control of a massive internet of things (IoT) enabled network of food retail refrigeration systems in proportion to the load available on the UK National Grid (i.e. an optimised Demand Side Response, DSR). The implementation utilises data from a large retail supermarket along with a purpose-built demonstration/experimental store (the 'barn') in our Riseholme Campus at the University of Lincoln.

## 2. Background

### 2.1. Deep Learning Frameworks

Deep Learning, and its ability to learn complex abstract representations from high dimensional data, is transforming the application of artificial intelligence. Deep learning can learn a new task from scratch, or adapt to perform tasks in various domains by transferring knowledge from one domain to another. Areas where DL has produced state-of-the-art results – and why we consider it here – are in particular; computer vision, and medical imaging [2, 3, 4, 5, 6]; signal, and natural language processing [7, 8, 9, 10]; agriculture [11, 12, 13]; and industry [14, 15, 16].

There are many popular single node deep learning frameworks currently available [17, 18, 19, 20]. However, there are fewer parallelisable deep learning frameworks [17, 21]. These can broadly be categorised by their parallelism strategies and capabilities, namely: [1]

- Model parallelism (MP) (TensorFlow, Microsoft CNTK), where a single DNN model is trained using a group of hardware instances, and a single data set (multi GPU in TensorFlow base case, multi-server in CNTK).

- Data parallelism (DP), where each hardware instance trains use across different data (usually separate models).

- Hybrid parallelism (HP) (Nemesyst), where a group of hardware trains a single model, but multiple groups can be trained simultaneously with independent data sets.

- Automatic selection (AP), where different parts of the training / inference process are tiled, with different forms of parallelism between tiles.

Of note, TensorFlow (core) is a model parallelism as it trains a single model over multiple GPUs using the same data. Distributed Tensorflow on the other hand is a form of hybrid parallelism as it creates its own MPI to orchestrate core TensorFlow between multiple machines; however – to best of our understanding – it cannot train/ use multiple distinct models between those machines simultaneously. As a consequence, the need for a hybrid parallelisable framework is not completely fulfilled for scenarios requiring multiple smaller separate models, for example where a model needs to be trained on individual machines in large aggregated networks, such as retail refrigerators (table 1). The Microsoft cognitive toolkit is similar to Distributed TensorFlow in that they are both frameworks for model parallelisation, but they lack the additional step that facilitates multiple and distributed simultaneous model training on the same or different data.

Nemesyst is a hybrid parallelism framework that can simultaneously train/infer across multiple groups of machines yielding bespoke models for each specific device. In addition, Nemesyst can utilise existing single node frameworks, such as Keras [20], since training scripts are separate and distinct. The Nemesyst codebase also includes examples and inbuilt functionality for multiple deep learning methods, such as RNN, LSTM, and GANs. Nemesyst's development was inspired, tested, and accommodated by the type of data and problems we encountered through the handling of real datasets. It represents a motion towards real-world unified pipelines that are robust and configurable for varying data, and the model handling involved in day-to-day operations of large industries and internet of things applications.

*2.2. Case Study - Demand Side Response Demonstration*

Here we demonstrate Nemesyst by optimising the energy use and system operation of a massive IoT network of retail refrigeration systems. High demands for energy, along with future commitments towards low carbon economies that include the generation of less predictable electrical loads (e.g. wind / solar renewables) is placing National Grid infrastructure under increased strain. To stabilise the Grid, focus is now placed on developing new approaches to manage load via the application of Demand Side Response (DSR) measures. During DSR events individual electrical demand across massive aggregated networks of machines is adjusted in response to available grid supply. Measures span financial incentives for commercial/industrial users to reduce demand during peak times when there is low available power and vice versa [22]. Many DSR incentives require the control of large loads and the aggregation of multiple industrial processes and machines [23, 24]. Standard static DSR mechanisms includes Firm Frequency Responses (FFR), where load is shed when the grid frequency drops to a predefined threshold. Figure 1 shows the two transitions typically encountered in a static FFR event. In the first transition (also known as primary FFR), industry has to shed load quickly (2 to 10s) and sustain this for 30 seconds; for the second transition the load can be held off for 30 minutes (also known as secondary FFR). Primary FFR's aim is to arrest frequency deviations from the nominal value, whereas secondary FFR is performed to contain and partially recover the frequency [25].

The application of DSR to massive networks of machines requires exceptional levels of coordinated control across individual devices. Food refrigeration systems can be deployed within DSR events since the thermal inertia / mass of food acts as a store of cold energy (effectively a battery). However, the thermal inertia in a retail refrigeration case will change as the food is actively shopped by consumers and then refilled, in addition ambient temperature can change, networks of stores can have multiple refrigeration systems and different foods will need bespoke control. This complexity provides an excellent opportunity for Nemesyst since deep learning is required to model thousands of assets simultaneously, with a need to continuously retrain multiple thousands of models / machine dynamics. The deployed control model must intelligently select which refrigerators to shed electrical load / turn
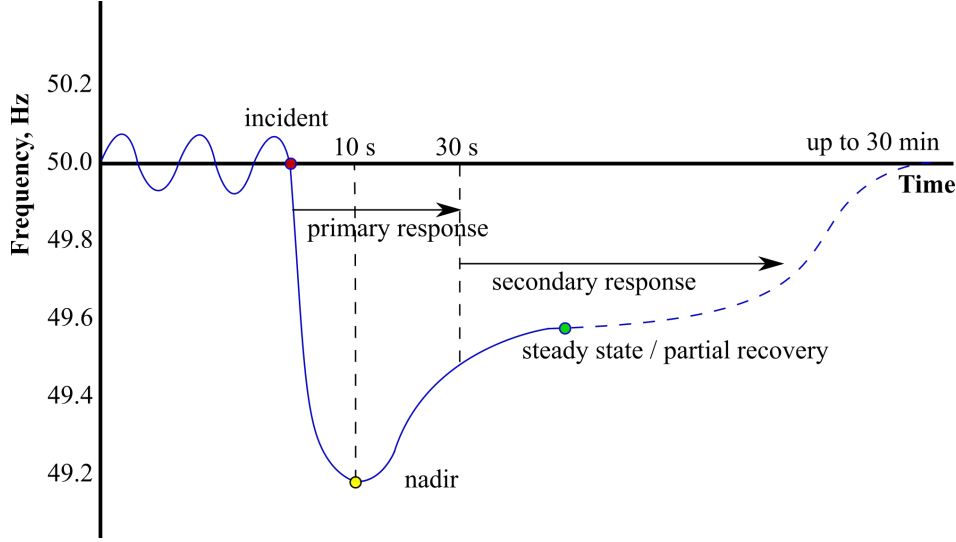
Figure 1: (colour online) Firm Frequency Response overview (National Grid).

off across a massive pool; one UK retailer alone operates >100,000 devices. This requires an algorithm that can predict the thermal inertia in individual cases and therefore how long they can be shut down (typically up to a maximum of up to 30 minutes) before the food temperature breaches a legally defined set point. Deep learning continuously re learns the thermal inertia of each device. To deliver DSR, a seperate candidacy algorithm selects which refrigerators can be shut down across an aggregated pool and for how long. For compliant DSR, a fixed electrical load, typically greater then 10MW, must be shed across the entire pool of machines for up to 30 minutes.

More generally, the application of deep learning for refrigeration system control is limited, for example Hoang [26] purports gains of 7% in daily chiller energy efficiency by using deep learning opposed to rote physical control models. There have also been some studies focussing on load forecasting for home appliances and energy disaggregation including refrigerators [27, 28, 29]. Although quite a few studies exist in related areas of research, such as demand response algorithms for smart-grid ready residential buildings in [30] or demand response strategies in autonomous micro-grids in [31], there has been no studies specifically deploying deep learning for DSR control in food retailing refrigeration systems.

## 3. Nemesyst Framework

The goal of this framework is to facilitate the use of deep learning on large distributed datasets in a distributable, accurate, and timely manner. As such, each stage must consider the following criteria:

- Handle data out-of-core as the incoming data may be too large to store in memory at any single instance

- Retrieve necessary data from locations that may differ to where the local framework process exists

- Disseminate models to the agents that require them for inference.

To achieve our goals and also be more agnostic and generalisable with respect to deep learning problems and techniques, we propose a framework primarily consisting of four core components (figure 2):
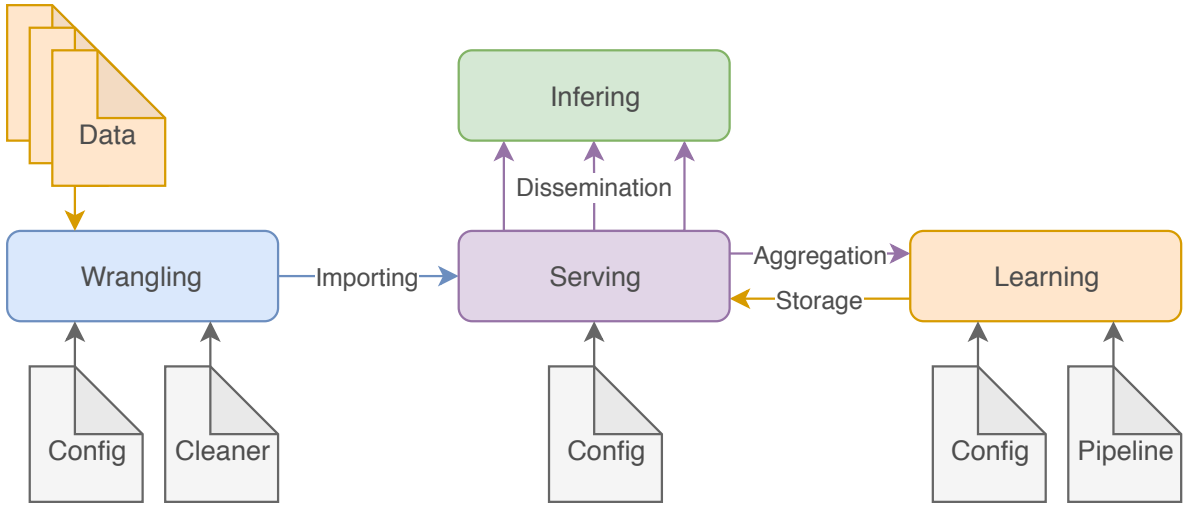
4

Figure 2: (colour online) The proposed high level framework architecture.

### 3.1. Wrangling

Wrangling; where the original data from a source is reformed into individual atomised/single usable examples, which will later be imported into a database such as MongoDb. This stage seeks to filter out the unusable data and to clean the remaining data from artifacts and inconsistencies.

As long as the data comprise single usable examples, e.g. in many-one sequence prediction multiple rows with a target column of a consistent value, in many sequence generation multiple rows with multiple values in the target column, etc., the data can be selectively chosen through MongoDB's aggregation pipelines. This approach allows for maximal flexibility and also generalises well, since a single example can be abstracted by the implementer via custom config and cleaning scripts. Since databases are not tailored specifically to deep learning and they are widely used, this forces us towards existing standards the benefit of which is that we can then utilise already existing frameworks for databases like Amazons DynamoDB for MongoDB and then further exploit the capabilities of these frameworks to manage our databases for use with in-place encryption and transport layer security (TLS).

### 3.2. Serving

Serving; where clean and atomised data exists and can be stored such that it is readily available for the learners to access at the point of need; and as a consequence of being stored in a database can now be properly indexed. Subsequently indexing the data using a database means we are now free to utilise the advanced functionality of these databases including the aggregation pipelines, sharding (critical to distribution), searching, sorting, filtering, finding/ querying, and replica sets (critical for up-time and availability).

When a database is used in this manner, i.e separate the data cleaning and data storage from any specific learning script, the user creates a consistent interface for the learning scrips. Unlike most deep learning scrips, our system is capable of branching many different and new algorithms as the server and learner are separate, this becomes the message passing interface (MPI). This stage also serves any trained models as learners are expected to return serialised binary objects of the learned weights and architecture.

### 3.3. Learning

Learning; where the aggregated data is fetched in rounds to train a learner in batches, at which point the learner returns the training loss, validation loss, neural network architecture (sequential and

binarised) to the serving database for dissemination for further learning, and use. At this stage it is possible and advisable to take advantage of other frameworks such as tensorflow, and possibly their distribute simultaneous training functionality on single machines. The learning stage should be highly and completely configurable as it should not matter how a model is trained as long as it is returning all components desired in future, such as the trained model, and/ or training parameters.

## 3.4. Architecture Consideration Towards Goals

We have implemented a form of this framework, dubbed: Nemesyst, which incorporates deep learning techniques, such as RNNs, LSTMs, and GANs, and demonstrated its applicability and impact on a large scale industrial application of high impact. We have also released the code for this framework implementation under a permissive licence and will continue to add to it at:
https://github.com/DreamingRaven/Nemesyst

## 4. Methods and Tools

For the sake of both completeness and to maintain compactness this shall only be a brief look at the most important tools used.

- Keras and Tensorflow;[20] De facto standard for production level deep learning, utilising graphics processing unit (GPU) parallelisation to speed up computation. Both are free/ open-source removing the need for expensive corporate licensing

- MongoDb; perhaps the most popular noSQL database type, being document oriented, scalable, eventually consistent, and most importantly free/ open-source, and fast [32]

## 4.1. MongoDB

In this framework we propose and use MongoDB as the data storage method. This comes with a plethora of consequences:

- Distribution; utilising a client-server model through MongoDB affords the possibility to process individual models for individual tasks from a central database of models and/or data. In this case training a broader model for all refrigerators, which is then re-trained/ fine tuned individually in a distributed manner for individual machines

- Handling of large/ sharded data; Using a database allows much more complex data storage and representation to be handled, sorted, and maintained

- Querying; MongoDB is likely more effective at querying than bespoke system, as it is established, widely used and prevalent, meaning more support is available from the community itself

- Integration; due to MongoDB being a popular database, it is likely that this framework's MongoDB handling will be able to integrate to existing systems

- Complexity; handling any external and potentially changing framework can mean that the changes could cause unhandled exceptions, and temporary down time, until the differences can be adapted to

- Security; MongoDB is more mature and has many more available and used security implementations, for a variety of different scenarios, as will likely be the case upon application of this framework. This adaptability to the security need is an important part of the application of deep learning that is frequently ignored. Examples of such are in-place encryption, and transport layer security integration.

- Adaptability; the MongoDB aggregate pipeline and its generic nature has been included in this framework. This improve the application breadth. We note though that there must always be a target feature/ ground truth to be predicted along with the data to use for prediction.

6

| MongoDB Document | Model Meta-Data | Epochs: | 20 |
| | | Time Stamp: | 2018-10-24 16:11:04.047 |
| | | ... | ... |
| | | Loss Metric: | MAE |
| | | Loss: | 0.01 |
| | Model Binary | Binary('gANja2VyYXMuZW5naW5lLnN lcXVlbnRpYWwKU2VxdWVudGlhbApx ACmBcQF9cQJYC...') | |

Figure 3: (colour online) MongoDB document consecutively split into abstraction hierarchy.

### 4.2. Time-Series Analysis

Given the sequential nature of the signals we are dealing with in this problem, it was intuitive to utilise Recurrent Neural Networks (RNN), such as Long-Short-Term Memory (LSTM). Their cells can formulate a non linear output $a^{[t]}$ based on both the input data $x^{[t]}$ at the current time step $t$, and the previous time-step activation $a^{[t-1]}$, as described in (1), where $h$ (i.e. hyperbolic tangent) is a non-linear activation function.

$$a^t = h(x^{[t]}, a^{[t-1]}) \tag{1}$$

LSTM is a very popular and successful RNN architecture composed of a memory cell, an input gate, an output gate and a forget gate. The cell is able to hold in memory values over arbitrary time intervals, whereas the three gates regulate the flow of information into and out of the cell. More specifically a cell receives an input and stores it for a period of time. Intuitively this is equivalent to applying the identity function ($f(x) = x$) to the input. It is evident that the derivative of the identity function is constant; therefore when an LSTM model is trained with backpropagation through time the gradient does not vanish. Backpropagation through time is a gradient-based technique employed to train certain types of RNN models, such as LSTMs [33].

A major problem that LSTMs circumvented over the traditional recurrent neural networks (RNN) has been exploding and vanishing gradients, which one encounters when training using hundreds of time steps. LSTMs are well suited to problems where preserving the memory of an event that occurred many time steps in the past is paramount for identifying patterns and hence informing better predictions. As outlined previously, the purpose of the present study is not demonstrating or arguing the superiority of deep LSTM models, but we deploy the approach to evidence the impact that a state-of-the-art intelligent system can have in retail refrigeration systems. This study is the first step towards developing a system that can work in the background for thousands of refrigerator packs and provide candidates for DSR, whilst adhering to the requirements of the National Grid.

As descirbed an LSTM model was adopted due to its capability of learning long term dependencies on data. The equations relative to LSTM networks follow, and the reader is invited to refer to the original paper [33] for further details.
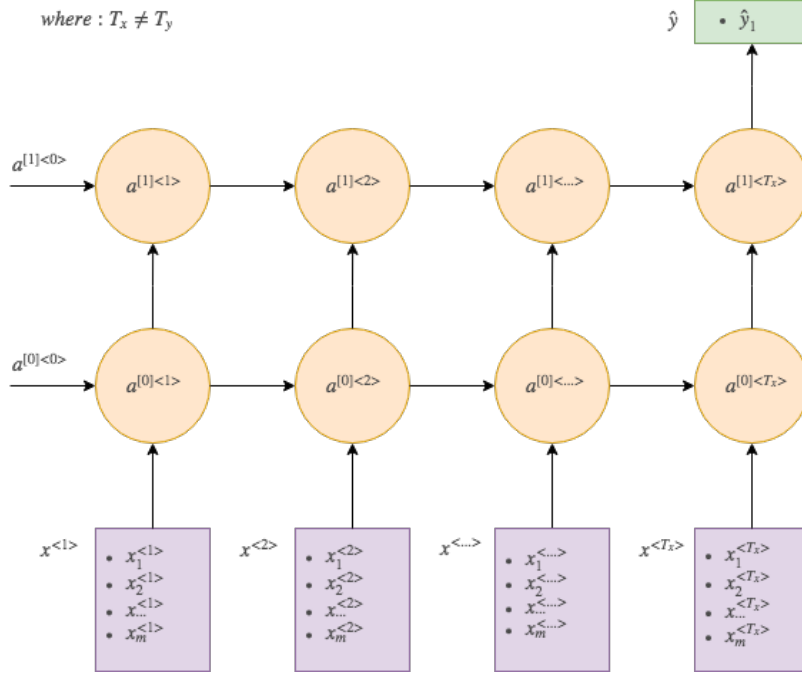
7

Figure 4: (colour online) Recurrent neural network architecture where: $x$ = single input example of a defrost, $x^{<n>}$ = the $n^{th}$ time step of this single example, $\hat{y}$ = is the output predictions (in this case only one per example), $a^{[n]<m>}$ = activation vector at layer n, and node/ time step m, $T_x$ = total number of time steps $T$ for examples $x$

$$
\begin{aligned}
\widetilde{c}^{[t]} &= tanh(W_{\widetilde{c}}[a^{[t-1]}, x^{[t]}] + b_{\widetilde{c}}) \\
\Gamma_u &= \sigma(W_u[a^{[t-1]}, x^{[t]}] + b_u) \\
\Gamma_f &= \sigma(W_f[a^{[t-1]}, x^{[t]}] + b_f) \\
\Gamma_o &= \sigma(W_o[a^{[t-1]}, x^{[t]}] + b_o) \\
c^{[t]} &= \Gamma_u \odot \widetilde{c}^{[t]} + \Gamma_f \odot c^{[t-1]} \\
a^{[t]} &= \Gamma_o \odot tanh(c^{[t]})
\end{aligned}
\tag{2}
$$

In 2, $c$ is the memory cell, $\Gamma_u$, $\Gamma_f$ and $\Gamma_o$ are the update, forget and output gates respectively; $W$ denotes the model's weights, and $b$ are bias vectors. These parameters are all jointly learned through backpropagation. Essentially, at each time-step, a candidate update of the memory cell is proposed (*i.e.* $\widetilde{c}^{[t]}$), and based on the gates $\Gamma$, $\widetilde{c}^{[t]}$ can be utilised to update the memory cell ($c^{[t]}$), and subsequently provide a non-linear activation of the LSTM cell ($a^{[t]}$). To learn a meaningful representation of the signals, two LSTMs were stacked, both with 400 neurons (figure 4).

*4.3. Generative Adversarial Network*

Generative Adversarial Networks (GANs) proposed by Goodfellow [34] are:

- adversarial models (competing/ co-operating neural networks)

- both trained simultaneously

- a generator (G) neural network which seeks to simulate the data distribution as closely as possible

- a discriminator (D) neural network which seeks to estimate the probability that any given sample came from the training data rather than simulated by G
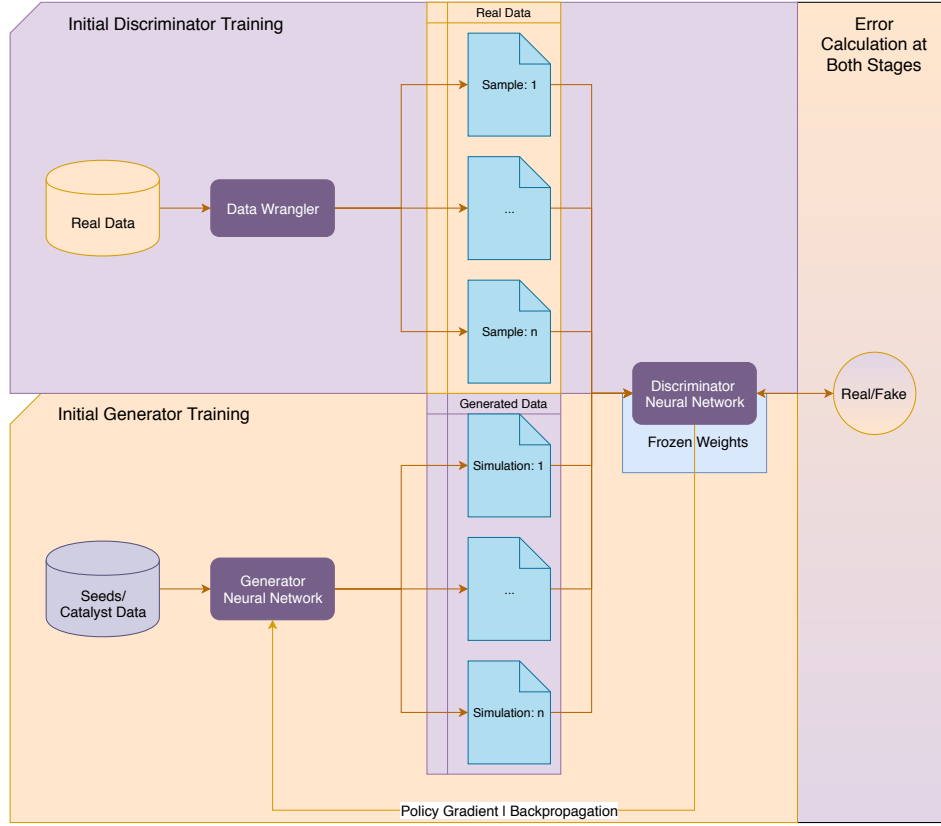
Figure 5: (colour online) General architecture/ flow of real and simulated data through a GAN.

GANs (figure 5) were conceptualised through game theory of a minmax two player game, where the generator and discriminator are trying to achieve what is called the Nash Equilibrium with respect to each other. In brief, the two components of GANs, i.e. generator and discriminator, are improving themselves by competing with each other. Generator starts with random noise as input; as the process evolves it arrives to a point where it has learned the underlying distribution of the input data, hence generating a very similar output. This process continues up until the generated data are indiscernible to the original ones. There have been advancements in the techniques used to train GANs, such as policy gradients, and the ability to predict discreet features for things, such as natural language processing or other sequence based tasks [35]. Using the variant of GANs called sequence-GAN or seqGAN, one could train these generative models on this predominantly sequence based/ time series prediction task.

   The principles of GANs are particularly relevant to our studies, given that they are simultaneously a type of generative and discriminative modelling compared to the discriminative nature of RNNs. As such GANs are very good at simulation, and provide an the opportunity to expand further into data correction, since the GAN model of learning seeks to mimic the data and its distribution, which was a large hindrance using this dataset. On the other hand, their primary drawbacks are that they take a more time to develop and are not at least in their vanilla form compatible with sequence data. [35]

## 5. Nemesyst for Intelligent refrigeration Systems

### 5.1. Case Study: Demand Side Response Recommendation

Demand side response (DSR) events simply modify the power demand load in proportion to available energym on the Grid [36, 37]. However, delivering DSR for supermarket refrigeration systems is complex, requiring high speed decisions on:

- Where to displace load with an aggregated number of machines. This requires accurate predictions/ recommendations across large pools of diverse machines

- Provisioning of data to train the algorithms for individual machines, in this instance the algorithm must learn the thermal inertia indirectly within each case to predict whether they can be safely shut down without a subsequent breech of food safety temperature limits during a DSR event; refrigerators with high thermal inertia can be shut down for a longer period than those with low inertia.

- Dealing with a large array of distributed assets, requiring a distributable algorithm or at least an algorithm that can handle database servers, which would likely be the only non-local place this data could be processed at point of need within tight time margins

- Time bound response, which requires both sufficient computational power and efficient algorithms to expedite the computation of this data to output within the required time-period

### 5.1.1. Data

Two sources of data were used to demonstrate Nemesyst, one from the experimental refrigeration research centre at The University of Lincoln (UoL; see for details [36, 37] and a second selected from a massive operational network (110,000 cases) of a national food retail company. Appendix B.1 shows the commercial stores' refrigerator operational data features, whereas Appendix B.2 shows a sample of the data gathered at UoL, which represents a single fridge and all associated sensors in the controlled test store.
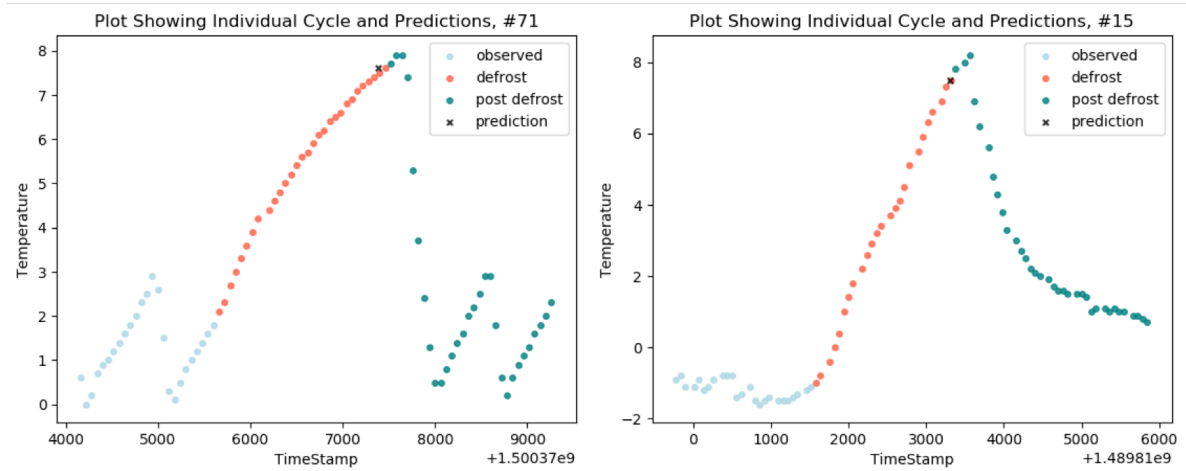


Figure 6: (colour online) Graph of air_on_temperature, against time, with overlayed prediction and colour coded sections for two fridges. Timestamp is simply the unix time since epoch. "Observed" (light blue) here means what the neural network is provided to learn from, "defrost" (orange) is the ground truth; of which "prediction" (x) is trying to predict the final ground truth (orange) value. The difference between the last "observed" (light blue), and final "defrost" (orange) value is the time (in seconds) that defrost can occur for before reaching the unsafe threshold of 8°c.

Our objective was to predict the time (in seconds) until the refrigerator temperature ("air_on_temperature") rises from the point it is switched off until it breaches a food safety threshold. This duration varies

between cases, for example the threshold is heuristically 8°c for high temperature refrigerators, and -183°c in low temperature refrigerators (freezers), however since we do not have all the information about the threshold for each fridge or even a good way of identifying the fridge types we train the neural network on the final point of defrost (see Fig 6) from the data as this is the most consistent point we can rely on. Subsequently the variation means a subset of fridges can be selected from the population which are capable of remaining cool through a DSR event. Thus knowing how long the refrigerators can remain off before reaching this threshold allows us to identify the best candidates which reduce the power consumption sufficiently enough, while also minimising the total number of affected refrigerators, and potentially affected stock. This can be visualised in figure 6 where the light blue observed data set is used to predict the duration of the defrost section (orange) before it reaches the unsafe threshold of 8 degrees Celsius (the difference between last light blue to the last orange) giving us a single predicted duration per refrigerator at any given point in time. If we used longer time periods we could potentially include information from previous defrosts but we found this unnecessary as the neural network was already more than capable with the mere fluctuations prior to defrost for prediction.

In the data from the large chain food retailer, the only indicator available on a consistent basis for how each fridge would act if it was switched off for DSR is its temperature response during a standard defrost cycle. These are routine and programmed 4 times a day for any given fridge. During a defrost cycle – although depending on the refrigerator – the fridge is turned off until it reaches a given temperature threshold. Case temperature data during the defrost cycles were used to train the neural networks as this was the most readily available information on refrigerator behavior when unpowered.

In the commercial data set, we used 110,000 defrost examples from 935 refrigerators (90,000 training, 10,000 validation, 10,000 testing), shared between 21 superstores after extraction. When optimised the data deployed by Nemesyst was 5.1KB per data set, which results in a total data scale of 550MB. We used only 3 months of data (limitation imposed by the data management process), and only for a subset of the 3000+ available stores i.e 21, while also eliminating redundancy and selecting certain portions of the available sequences. This will be significantly larger in practice, but is already too large to be visualised and traverse easily, another advantage of breaking this data down into usable atomic examples. The extracted valid sequences are on average 31.48 time-steps of c.1 minute long (not including missing/ empty time steps). This means we have roughly 3,462,800 time steps, each with a minimum of 7 valid data points per time step, resulting in 24,239,600 total data points processed for and subsequently used by a subset for any given neural network pre epoch. Although it should be noted an added difficulty in this scenario is that the overwhelming majority of the data is not sufficient in length or is problematic leaving on a subset that needs to be extracted for use in training. Lastly if we imagine this system will eventually be used at all stores, a simple system isolated to one machine in one store will not be sufficient to serve all stores and infer their data simultaneously. This shows the clear rationale for deploying distributing deep learning to internet of things systems, for both training and inference.

Data quality for the commercial stores data set posed significant problems, some of the these included:

- Missing data (features and observations). Missing features are automatically removed by Nemesyst, as they are easy to spot if the number of unique values of x are $<= 1$. Missing observations are dealt with in a customisable cleaner file in the wrangling stage as it may be desirable to maintain these observations in some form.

- Inconsistent data values, such as Yes, yes, y, YES etc, due to humans' inconsistent input. These are automatically dealt with at the unifying stage of the customisable cleaner file.

- Inconsistent number of sensors between stores and fridges, some having more and others having other inputs, making it more difficult to create a subset that share the same features/characteristics. This is dealt with in the customisable cleaner, as what is useful cannot be known and hence being able to deal with this automatically for all data.

$$MAE = \frac{\sum\limits_{i=0}^{n-1} |y_i - \hat{y_i}|}{n} \tag{3}$$

Mean Absolute Error (MAE), where:

- $y_i$ is the $i$'th target value, and $\hat{y_i}$ is the corresponding predicted value

- $|y_i - \hat{y_i}| = |e_i|$, is the absolute difference between the actual $y_i$ and predicted $\hat{y_i}$, otherwise known as $|e_i|$

- $n$ is the total number of ground truth target values, and the total number of predicted values.

- Duplicate entries, which is an expensive, time consuming operation to fix, but dealt with in the customisable cleaner.

- Extreme values, such as refrigerators at $50.0\,°C$ and below $-40.0\,°C$, can be dealt with by sigma clipping in the customisable cleaner.

Nemesyst requires data to be separated into single/ individual examples, so that they can be imported into the MongoDB database. It is then possible to use database functionality to both parallelise data handling, and unify the MPI of the system. As a side note, MongoDB has many tools built around it, in particular helpful visualisation tools that help speed up data exploration (figure A.1), and testing of aggregation pipelines (figure A.2).

*5.1.2. Nemesyst Configuration and Methodology*

In our public github repository, we show the implementation and source code of the RNNs, LSTMs, and GANs implemented in this study [38]. We also show the associated RNN architecture shown in figure 3, which can show the configurability of Nemesyst through simple adaptations in arguments to shift the entire architecture. It has been further extended to other techniques, such as GANs, which are included in our implementations. It can be extended to any other novel techniques due to the separation of the learner and Nemesyst core.

In our test case, we split the 110,000 defrost examples such that c.10,000 were used for final testing c. 10,000 were randomly selected each time for validation (Monte Carlo with replacement). We used a two-layer RNN and LSTM, using MAE (3) as our error function and adam [39, 40] as our optimiser. We chose MAE for this regression problem as the errors significance is equal, meaning we prefer to optimise the average case rather than penalise extreme cases. We chose the adam optimiser for our neural networks gradient descent as it is a lightweight optimiser with a lower training cost than, AdaDelta, RMSProp, etc. The GAN contains a generator consisting of 3 layers of RNN, leaky ReLU pairs [41], batch normalisation [42], and 0.8 nesterov momentum [43, 44]. Batch normalisation is simply the normalisation of parameter updates, between a group of examples, such as a group of size 32 will only result in a single parameter update that is some function of their constituent errors. The sequence length we settled for led to the greatest predictive power at 120 time-steps long. For the purposes of speed, we utilised our hybrid parallelisation using 3 different machines to train different models simultaneously, producing models in less than 2 minutes of training.

*5.1.3. Results and Discussion*

Table 1 shows the results achieved across the real and test stores, using all aforementioned models (bearing in mind that the data flow at a granularity of 1 minute). Single store refers to the store which returned the best performance in terms of MAE whereas All stores is the global model using data from all the stores available. It is important to note that the error is the mean absolute error (MAE) in seconds of predicted time until unsafe temperature is reached. Assuming the ground truth value

| Model Type | Data Source | Loss Train MAE (Epochs) | Loss Test MAE | Loss Ahead 2min (Test Set) | Training Data Points | Test Data Points |
|---|---|---|---|---|---|---|
| Vanilla RNN | Test Site | 64 (20) | 85 | - | 8640 | 8256 |
| Vanilla RNN | Single Store | 336 (23) | 640 | - | 11040 | 8256 |
| Vanilla RNN | All Stores | 601 (3) | 1418 | - | 84360 | 11520 |
| LSTM | Test Site | <10 (13) | <10 | 23 | 8640 | 8256 |
| LSTM | Single Store | 87 (13) | 119 | 224 | 11040 | 8256 |
| LSTM | All Stores | 286 (2) | 330 | 338 | 84360 | 11520 |
| seqGAN | Test Site | <10 (16) | <10 | 23 | 8640 | 8256 |
| seqGAN | Single Store | <10 (17) | 33 | 33 | 11040 | 8256 |
| seqGAN | All Stores | <10 (13) | 59 | 15 | 84360 | 11520 |

Table 1: Table showing results for Nemesyst applied to demand side response recommendation, error is seconds from now until unsafe threshold is met. This table also shows the results for future prediction, i.e. 2 minutes ahead of potential dsr event.
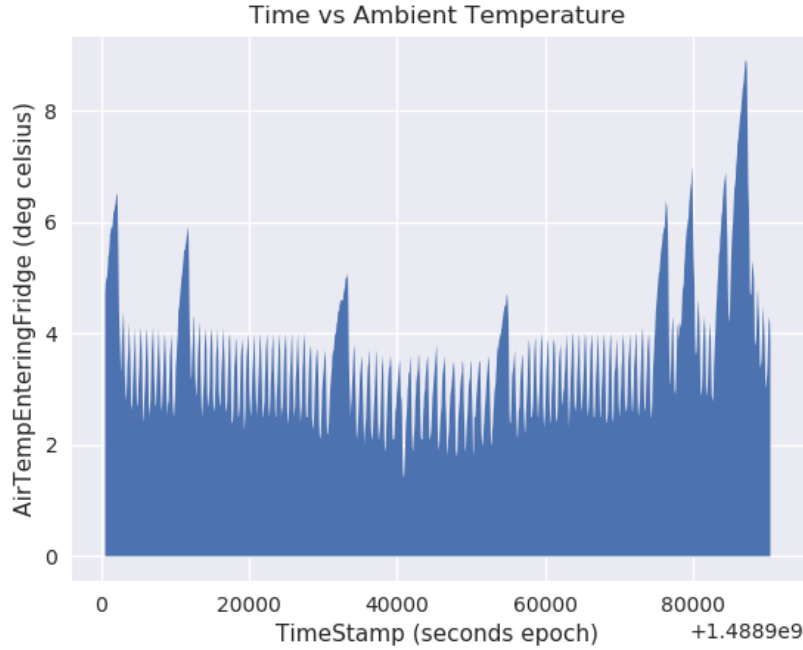


Figure 7: (colour online) Graph of air_on_temperature, against time for a full day.

of the dependent variable time (to unsafe threshold) is 1800, then we can report that one can predict this time $1800s \pm 10s$. It is worth pointing out that a normal defrosting cycle lasts between 1800 and 2700 seconds. That helps with placing the performance – in terms of MAE – of the presented models in Table 1 into perspective.

As can be seen an expected superiority in the performance of LSTM versus vanilla RNN occur, which is down to the long dependencies on historical data, in particular since the data of any single refrigerator has instances, denoting when they have been turned off/defrosting before (example can be seen in figure 7) . In table 1 we have also listed our future predictions with our developed LSTM, and GAN that show how predicting for a DSR event two minutes ahead has a minimal impact on results. In table 2, single store performance (33 seconds in terms of MAE, using GAN) demonstrates clearly how feasible and plausible deploying such a system – error-wise – would be. Even using a global model for all stores the performance of 128 seconds in terms of MAE, suggests that being able to predict with such a low MAE, at 60 seconds prior to a defrosting cycle kicking in is a promising indicator. To this direction the performance we achieved in the barn store (<10 and 12 seconds in terms of MAE for in time and 2 min ahead predictions respectively) is a good reference point of what the absolute
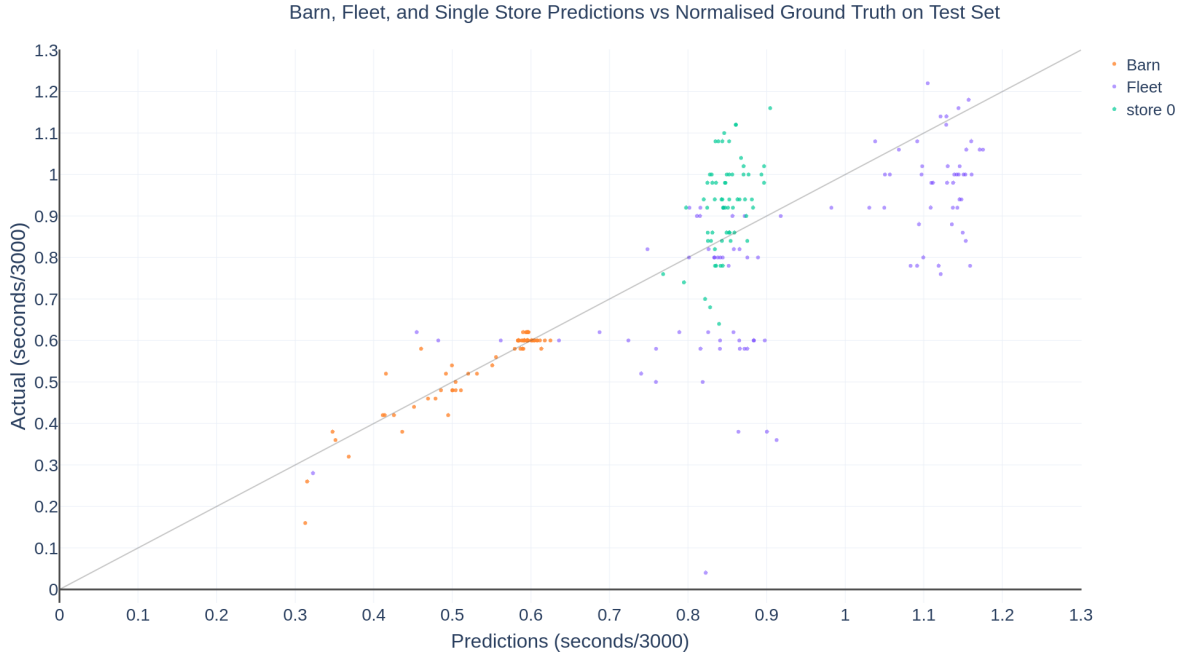
Figure 8: (colour online) Actual versus Predicted values plot, showing how close to the reference (diagonal) line the predictions are (barn, whole fleet and a random store highlighted in a different colour). Any points falling on the diagonal line denote perfect predictions. It's worth pointing out that the predictions/dots in this plot (based on a subset of the test data) exceed the 200 ones; but many of them overlap with each other on the same spot. It is evident that the barn model performs better as can be seen in 1, the reasons for this are explained in the main text. However, both models visually demonstrate how promising the results are towards introducing an intelligent system for DSR.

performance under a controlled environment can be.

Table 1 clearly demonstrates that training models on individual or a more tailored store specific data sets provides superior performance compared to global "all stores" data. This suggests that distributing the learning to more specific learners would be more beneficial, in particular since to train a global "all stores" model restricts the model to using only the subset of features shared between all the stores, as opposed to the more available data in one store that might not be consistent enough to use across all of them. Besides, each store exposes an underlying distinct behaviour, such as number of visitors, preference in fresh versus other products, etc., which means that a bespoke individual model for each store – substantiated by the results provided as well – might be the answer.

The results also show a significant difference between the test site (barn) and the real sites (as evidenced by figure 8 as well). Authors believe this could be due to two reasons. First, the test site has some limitations in terms of how much real-life behaviour and shopping patterns can be captured, albeit some interventions, such as opening and closing the fridges, were incorporated in the data utilised. Second and probably most importantly, the availability of complete and reliable datasets allow for a more concrete representation of the principles governing the operation of the fridges, allowing long-term patterns to be reliably identified and exploited. That is a very crucial message this paper aims at conveying, i.e. improving the availability of sensor data, investing on data science principles, such as data consistency/fidelity, establishing quality control processes, flagging of intermittent transmission of information, and generating missing values in (near) real-time, are factors that will make a massive difference in the reliability and performance of intelligent systems.

To the best of our knowledge, this is the first study to show how an intelligent system based on a bespoke framework that can automatically handle and pre-process data, coupled with deep learning techniques can be incorporated in the workflow of IoT machine optimisation This was demonstrated for large retail supermarkets to optimise operational processes, decision-making process and subsequently

reduce operational energy costs.

## 6. Conclusions and Future Work

This paper proposed a new deep learning-based framework, i.e. Nemesyst which can provide a solution and unification of deep learning techniques using a common interface, as well as allowing hybrid parallelisation and operate within an IoT architecture. Nemesyst can store and retrieve generated models (weights, architecture), model data (train, test validation) and model metadata. As part of this work, the framework was applied to demand side response recommendation. Our proposed framework demonstrated state-of-the-art results on predicting time until fridge temperature rises to a set point (in time and also 2 minutes ahead of defrosting cycle), which is directly applicable to demand side response recommendation in refrigeration systems. It has also been shown via comparisons between the main fleet and the barn store how slight variations in data availability and consistency can have a significant effect on the learning performance, suggesting a direct improvement in predictive performance, should data availability and fidelity be improved in the future as part of the day-to-day data aggregation process. This claim has been substantiated by several working examples using state-of-the-art models, such as LSTMs and GANs, which demonstrated that such a framework and models can be deployed to improve the scheduling of the defrosting cycles towards reacting to DSR. Lastly we have introduced Nemesyst, and released it openly as discussed previously so that others may use, and modify it freely, potentially contributing and furthering the project.

## Acknowledgements

## Conflicts of Interest

Authors have no conflict of interest to declare.

## References

[1] Dhabaleswar K DK Panda, Hari Subramoni, and Ammar Ahmad Awan. High performance distributed deep learning: A beginners guide. In *Proceedings of the 24th Symposium on Principles and Practice of Parallel Programming*, pages 452–454. ACM, 2019.

[2] Andre Esteva, Alexandre Robicquet, Bharath Ramsundar, Volodymyr Kuleshov, Mark DePristo, Katherine Chou, Claire Cui, Greg Corrado, Sebastian Thrun, and Jeff Dean. A guide to deep learning in healthcare. *Nature medicine*, 25(1):24, 2019.

[3] Dimitrios Kollias, Miao Yu, Athanasios Tagaris, Georgios Leontidis, Andreas Stafylopatis, and Stefanos Kollias. Adaptation and contextualization of deep neural network models. In *2017 IEEE Symposium Series on Computational Intelligence (SSCI)*, pages 1–8. IEEE, 2017.

[4] Fabio De Sousa Ribeiro, Liyun Gong, Francesco Calivá, Mark Swainson, Kjartan Gudmundsson, Miao Yu, Georgios Leontidis, Xujiong Ye, and Stefanos Kollias. An end-to-end deep neural architecture for optical character verification and recognition in retail food packaging. In *2018 25th IEEE International Conference on Image Processing (ICIP)*, pages 2376–2380. IEEE, 2018.

[5] Siqi Nie, Meng Zheng, and Qiang Ji. The deep regression bayesian network and its applications: Probabilistic deep learning for computer vision. *IEEE Signal Processing Magazine*, 35(1):101–111, 2018.

[6] Fabio De Sousa Ribeiro, Francesco Caliva, Mark Swainson, Kjartan Gudmundsson, Georgios Leontidis, and Stefanos Kollias. Deep bayesian uncertainty estimation for adaptation and self-annotation of food packaging images. *arXiv preprint arXiv:1812.01681*, 2018.

[7] Hao Ye, Geoffrey Ye Li, and Biing-Hwang Juang. Power of deep learning for channel estimation and signal detection in ofdm systems. *IEEE Wireless Communications Letters*, 7(1):114–117, 2018.

[8] Qingchen Zhang, Laurence T Yang, Zhikui Chen, and Peng Li. A survey on deep learning for big data. *Information Fusion*, 42:146–157, 2018.

[9] Fabio De Sousa Ribeiro, Francesco Calivá, Dionysios Chionis, Abdelhamid Dokhane, Antonios Mylonakis, Christophe Demaziere, Georgios Leontidis, and Stefanos Kollias. Towards a deep unified framework for nuclear reactor perturbation analysis. In *2018 IEEE Symposium Series on Computational Intelligence (SSCI)*, pages 120–127. IEEE, 2018.

[10] Tom Young, Devamanyu Hazarika, Soujanya Poria, and Erik Cambria. Recent trends in deep learning based natural language processing. *ieee Computational intelligenCe magazine*, 13(3):55–75, 2018.

[11] Andreas Kamilaris and Francesc X Prenafeta-Boldu. Deep learning in agriculture: A survey. *Computers and Electronics in Agriculture*, 147:70–90, 2018.

[12] Jayme Garcia Arnal Barbedo. Plant disease identification from individual lesions and spots using deep learning. *Biosystems Engineering*, 180:96–107, 2019.

[13] John Ringland, Martha Bohm, and So-Ra Baek. Characterization of food cultivation along roadside transects with google street view imagery and deep learning. *Computers and Electronics in Agriculture*, 158:36–50, 2019.

[14] Qingchen Zhang, Laurence T Yang, Zheng Yan, Zhikui Chen, and Peng Li. An efficient deep learning model to predict cloud workload for industry informatics. *IEEE Transactions on Industrial Informatics*, 14(7):3170–3178, 2018.

[15] Liangzhi Li, Kaoru Ota, and Mianxiong Dong. Deep learning for smart industry: efficient manufacture inspection system with fog computing. *IEEE Transactions on Industrial Informatics*, 14(10):4665–4673, 2018.

[16] Qingchen Zhang, Laurence T Yang, Zhikui Chen, and Peng Li. A tensor-train deep computation model for industry informatics big data feature learning. *IEEE Transactions on Industrial Informatics*, 14(7):3197–3204, 2018.

[17] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[18] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. Caffe: Convolutional architecture for fast feature embedding. *arXiv preprint arXiv:1408.5093*, 2014.

[19] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. *NIPS*, 2017.

[20] François Chollet et al. Keras: The python deep learning library. *Astrophysics Source Code Library*, 2018.

[21] Frank Seide and Amit Agarwal. Cntk: Microsoft's open-source deep-learning toolkit. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 2135–2135. ACM, 2016.

[22] Peter Bradley, Alexia Coke, and Matthew Leach. Financial incentive approaches for reducing peak electricity demand, experience from pilot trials with a uk energy provider. *Energy Policy*, 98:108–120, 2016.

[23] Ramon Granell, Colin J Axon, David CH Wallom, and Russell L Layberry. Power-use profile analysis of non-domestic consumers for electricity tariff switching. *Energy Efficiency*, 9(3):825–841, 2016.

[24] Philipp Grünewald and Jacopo Torriti. Demand response from the non-domestic sector: Early uk experiences and future opportunities. *Energy Policy*, 61:423–429, 2013.

[25] Fei Teng, Marko Aunedi, Danny Pudjianto, and Goran Strbac. Benefits of demand-side response in providing frequency response service in the future gb power system. *Frontiers in Energy Research*, 3:36, 2015.

[26] Hoang Dung Vu, Kok-Soon Chai, Bryan Keating, Nurislam Tursynbek, Boyan Xu, Kaige Yang, Xiaoyan Yang, and Zhenjie Zhang. Data driven chiller plant energy optimization with domain knowledge. *CoRR*, abs/1812.00679, 2018.

[27] Weicong Kong, Zhao Yang Dong, David J Hill, Fengji Luo, and Yan Xu. Short-term residential load forecasting based on resident behaviour learning. *IEEE Transactions on Power Systems*, 33(1):1087–1088, 2018.

[28] Peiyang Guo, Jacqueline CK Lam, and Victor OK Li. Drivers of domestic electricity users price responsiveness: A novel machine learning approach. *Applied energy*, 235:900–913, 2019.

[29] Eklas Hossain, Imtiaj Khan, Fuad Un-Noor, Sarder Shazali Sikander, and Md Samiul Haque Sunny. Application of big data and machine learning in smart grid, and associated security concerns: A review. *IEEE Access*, 7:13960–13988, 2019.

[30] Fabiano Pallonetto, Mattia De Rosa, Federico Milano, and Donal P Finn. Demand response algorithms for smart-grid ready residential buildings using machine learning models. *Applied Energy*, 239:1265–1282, 2019.

[31] Paraskevas Panagiotidis, Andrew Effraimis, and George A Xydis. An r-based forecasting approach for efficient demand response strategies in autonomous micro-grids. *Energy & Environment*, 30(1):63–80, 2019.

[32] Elif Dede, Madhusudhan Govindaraju, Daniel Gunter, Richard Shane Canon, and Lavanya Ramakrishnan. Performance evaluation of a mongodb and hadoop platform for scientific data analysis. In *Proceedings of the 4th ACM workshop on Scientific cloud computing*, pages 13–20. ACM, 2013.

[33] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[34] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014.

[35] Lantao Yu, Weinan Zhang, Jun Wang, and Yong Yu. Seqgan: Sequence generative adversarial nets with policy gradient. *CoRR*, abs/1609.05473, 2016.

[36] Ibrahim Saleh, Andrey Postnikov, Chris Bingham, Ronald Bickerton, Argyrios Zolotas, Simon Pearson, et al. Aggregated power profile of a large network of refrigeration compressors following ffr dsr events. *International Conference on Energy Engineering*, 2018.

[37] Ibrahim Saleh, Andrey Postnikov, Corneliu Arsene, Argyrios Zolotas, Chris Bingham, Ronald Bickerton, and Simon Pearson. Impact of demand side response on a commercial retail refrigeration system. *Energies*, 11(2):371, 2018.

[38] George Onoufriou. Nemesyst lstm. `https://github.com/DreamingRaven/Nemesyst/blob/master/examples/lstm.py`, 2019.

[39] Diederik P. Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization. *arXiv e-prints*, page arXiv:1412.6980, Dec 2014.

[40] Sashank J. Reddi, Satyen Kale, and Sanjiv Kumar. On the convergence of adam and beyond. In *International Conference on Learning Representations*, 2018.

[41] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[42] Sergey Ioffe and Christian Szegedy. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. *arXiv e-prints*, page arXiv:1502.03167, Feb 2015.

[43] Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton. On the importance of initialization and momentum in deep learning. In *International conference on machine learning*, pages 1139–1147, 2013.

[44] Timothy Dozat. Incorporating nesterov momentum into adam. 2016.

# Appendix A. Figures



Figure Appendix A.1: This figure shows MongoDB compass automatically analysing the schema for data exploration, and highlighting areas of interest, in particular here extreme temperature values that may need to either be explained or clipped.
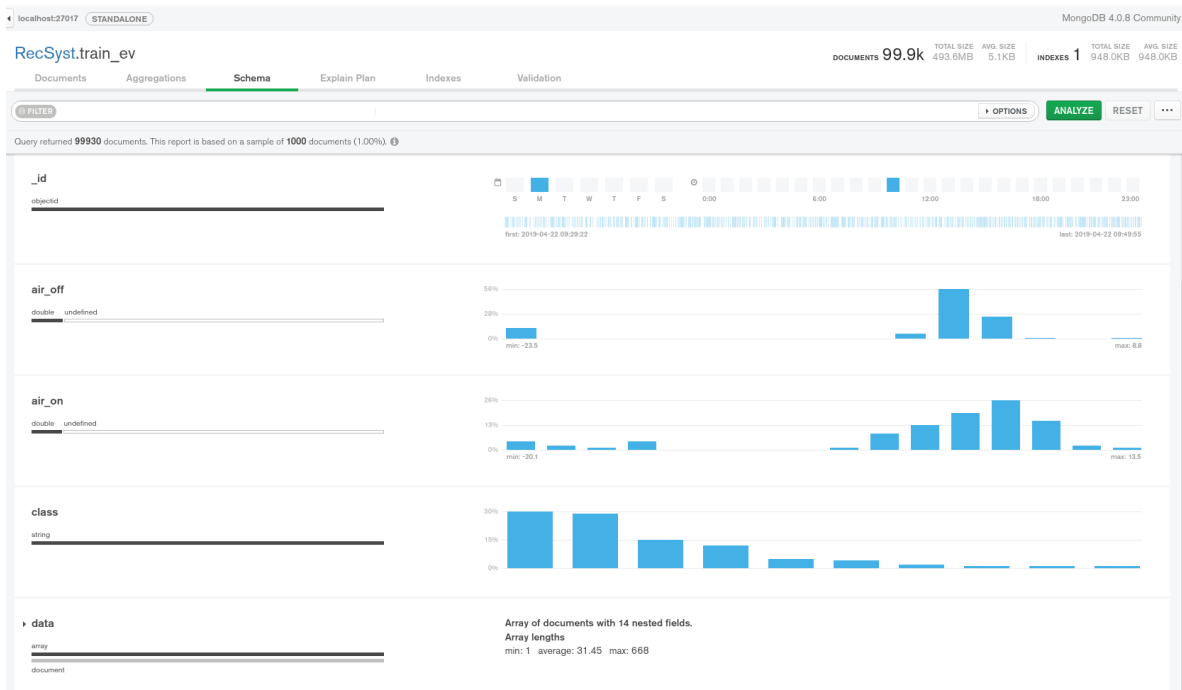


Figure Appendix A.2: This figure shows MongoDB compass automatically analysing the schema for data exploration, and highlighting areas of interest, in particular an in-balanced training set, which then had to be balanced.

Figure Appendix A.3: Example image of MongoDB aggregate pipeline testing, and output.

# Appendix B. Tables

| Feature | Barn | Fleet |
|---|---|---|
| defrost_state | 1 | 1 |
| air_on_temperature | 1 | 1 |
| air_off_temperature | 1 | 1 |
| name | 1 | 1 |
| refrigeration_case | 1 | 1 |
| class | 1 | 1 |
| severity | 1 | 1 |
| state | 1 | 1 |
| ip | 0 | 1 |
| store_name | 0 | 1 |
| store_number | 0 | 1 |
| timestamp | 1 | 1 |
| derived_shelfTemp | 1 | 0 |
| derived_foodTemp | 1 | 0 |
| evaporator_in | 1 | 0 |
| evaporator_out | 1 | 0 |
| evaporator_valve | 1 | 0 |
| door_state | 1 | 0 |
| date | d | d |
| targetTemp_on | d | d |
| targetTemp_on_diff | d | d |
| targetTemp_off | d | d |
| targetTemp_off_diff | d | d |
| targetTime | d | d |
| targetTime_sec | d | d |
| timestamp_sec | d | d |
| time_diff_sec | d | d |

Table Appendix B.1: Table showing feature prevalence between data sets, where d is derived, 1 is present and 0 is absent.

| TimeStamp | air_on | air_off | foodTmp | shelfTmp | Def | Ctrl | dSP | EvIn | EvOut | EvVlv | Def1 | Cln | Door | Lgt | Is | Is1 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1488990480.0 | 3.8 | 1.7 | 3.8 | 2.5 | 17.2 | 1.7 | -0.3 | -9.4 | 3.5 | 100.0 | 0 | 0 | 0 | 1 | 1 | 0 |
| 1488990420.0 | 4.2 | 2.7 | 3.8 | 3.3 | 17.1 | 2.7 | 0.7 | -9.0 | 3.7 | 100.0 | 0 | 0 | 0 | 1 | 1 | 0 |
| 1488990300.0 | 4.3 | 3.2 | 3.8 | 3.6 | 17.2 | 3.2 | 1.2 | -7.2 | 3.5 | 100.0 | 0 | 0 | 0 | 1 | 1 | 1 |
| 1488990240.0 | 4.1 | 2.8 | 3.8 | 3.3 | 17.2 | 2.8 | 0.8 | -3.9 | 3.2 | 0.0 | 0 | 0 | 0 | 1 | 0 | 0 |
| 1488990180.0 | 3.8 | 2.3 | 3.8 | 2.9 | 17.3 | 2.3 | 0.3 | -5.0 | 3.0 | 0.0 | 0 | 0 | 0 | 1 | 0 | 0 |
| 1488990120.0 | 3.5 | 1.8 | 3.9 | 2.4 | 17.3 | 1.8 | -0.2 | -4.8 | 2.8 | 0.0 | 0 | 0 | 0 | 1 | 0 | 0 |
| 1488990060.0 | 3.3 | 1.1 | 3.9 | 1.9 | 17.3 | 1.1 | -0.9 | -6.0 | 2.7 | 0.0 | 0 | 0 | 0 | 1 | 0 | 0 |
| 1488989940.0 | 3.1 | 0.5 | 3.9 | 1.5 | 17.4 | 0.5 | -1.5 | -8.1 | 2.6 | 0.0 | 0 | 0 | 0 | 1 | 0 | 0 |
| 1488989880.0 | 3.0 | 0.1 | 3.9 | 1.2 | 17.5 | 0.1 | -1.9 | -7.3 | 2.6 | 0.0 | 0 | 0 | 0 | 1 | 0 | 0 |
| 1488989820.0 | 3.1 | -0.1 | 4.0 | 1.1 | 17.5 | -0.1 | -2.1 | -8.3 | 2.7 | 0.0 | 0 | 0 | 0 | 1 | 0 | 0 |
| 1488989760.0 | 3.5 | 0.8 | 4.0 | 1.8 | 17.4 | 0.8 | -1.2 | -8.6 | 3.2 | 0.0 | 0 | 0 | 0 | 1 | 0 | 0 |
| 1488989700.0 | 3.9 | 1.6 | 4.0 | 2.5 | 17.4 | 1.6 | -0.4 | -8.5 | 3.5 | 100.0 | 0 | 0 | 0 | 1 | 1 | 0 |
| 1488989580.0 | 4.3 | 2.4 | 4.0 | 3.1 | 17.3 | 2.4 | 0.4 | -9.6 | 3.8 | 100.0 | 0 | 0 | 0 | 1 | 1 | 1 |
| 1488989520.0 | 4.5 | 3.1 | 4.1 | 3.6 | 17.4 | 3.1 | 1.1 | -7.6 | 3.7 | 100.0 | 0 | 0 | 0 | 1 | 1 | 1 |
| 1488989460.0 | 4.3 | 2.9 | 4.1 | 3.4 | 17.2 | 2.9 | 0.9 | -4.3 | 3.4 | 0.0 | 0 | 0 | 0 | 1 | 0 | 0 |
| 1488989400.0 | 4.0 | 2.4 | 4.1 | 3.0 | 17.1 | 2.4 | 0.4 | -4.9 | 3.3 | 0.0 | 0 | 0 | 0 | 1 | 0 | 0 |
| 1488989340.0 | 3.7 | 1.6 | 4.1 | 2.4 | 16.8 | 1.6 | -0.4 | -5.7 | 3.0 | 0.0 | 0 | 0 | 0 | 1 | 0 | 0 |

Table Appendix B.2: Sample of partially cleaned data from refrigerator operation (BARN)

| Feature | Barn | Fleet |
|---:|---|---|
| refrigeration_case | 0 | 1 |
| store_number | 0 | 1 |
| timestamp | 0 | 1 |
| fault_name | 0 | 1 |

Table Appendix B.3: Table showing feature prevalence between data sets for work order faults, where d is derived, 1 is present and 0 is absent.