ALMA MATER STUDIORUM - UNIVERSITÀ DI BOLOGNA

SCHOOL OF ENGINEERING AND ARCHITECTURE

Department of Electrical, Electronic, and Information Engineering "Guglielmo Marconi" - DEI

TELECOMMUNICATION ENGINEERING

MASTER THESIS

in

Mobile Radio Networks

PROBABILISTIC REGRESSION AND ANOMALY DETECTION FOR LATENCY ASSESSMENT IN MOBILE RADIO NETWORKS

Filippo Antonini

Supervisor:

Prof. Roberto Verdone

Co-Supervisor:

Ing. Marco Skocaj

Academic year 2022/23

Sommario

1. Introduction

One of the prominent features of 5G is the flexibility: from the different available physical layer configurations (many available formats of time slots and antenna configurations) to the different kinds of application provided through network slicing. One of the pillars driving the design and evolution of flexible next-generation 5G systems is the inclusion of autonomous and proactive radio resource management capabilities. To this end, the concept of Zero-touch networks aims to eliminate the need for costly and inefficient human intervention in network management procedures. Within such type of networks, estimating the Quality of Service (QoS) in advance becomes essential. Predictive QoS (PQoS) involves using advanced analytics and machine learning techniques to anticipate network conditions, traffic patterns, and user demands. By analysing historical and real-time data, predictive models can estimate network performance metrics like latency, throughput, and packet loss.

In particular, there are some new applications, like V2X, that require fast and precise forecasting of the latency made at the mobile network edge [1]. As will be shown in the subsequent section, a plethora of forecasting solutions have been extensively examined in recent years.

This work offers a comprehensive analysis and experimental findings of ML-based predictive latency, with a particular focus on the task of probabilistic regression and anomaly detection. The models developed for this purpose can serve as a valuable instrument for network designers seeking to analyse the correlations between specific network Key Performance Indicators and latency.

The experimental results presented in this work are:

- A Machine learning—aided selection of the main Key Performance Indicators influencing the latency.
- The comparison between a probabilistic regression and a probabilistic Bayesian regression of the latency
- The comparison between different architectures of autoencoders and variational autoencoders for the anomaly detection of the latency

This work starts by illustrating the state of the art in latency prediction and anomaly detection, then the KPIs and algorithms used are explained, concluding with the obtained results.

2. State of the art

Several studies focus on the implementation of ML learning and deep learning-based PQoS within mobile radio networks.

Many researches utilize ML algorithms and fully-connected neural networks: a linear regression model and a dense neural network are implemented in [2] to predict the throughput in V2X. Still in V2X, linear regression, multilayer perceptron and random forest are used in a similar scenario in [3], but the target metric is the latency; an instantaneous regression of the latency is performed in [9], leveraging a probabilistic Bayesian neural network. The advantages and limits of Bayesian and robust Bayesian learning applied in wireless networks are analysed in [8].

Other studies focus on convolutional neural networks (CNN) and recurrent neural networks (RNN) [4]. CNNs leverage compact convolutional kernels to capture localized information within a restricted receptive field, rather than relying on fully connected layers. RNNs are models designed to capture the dependencies in a time series, so they are widely used for forecasting.

Additional studies [4] perform spatial–temporal predictions by combining CNNs and RNNs or through Graph Neural Networks [9].

There are several techniques that have been used to detect anomalies, from probabilistic neural networks to PCA and autoencoders. An autoencoder is leveraged in [9]. According to [5] finding an appropriate scoring function is as important as finding an optimal algorithm. Authors in [6] develop a variational autoencoder model, achieving remarkable performance.

3. System model

Key Performance Indicators (KPIs) are metrics used by the mobile network operators to assess and measure the performance and quality of the network. In this study, the initial phase involves constructing a dataset comprising the most relevant KPIs associated with latency in the feature space, alongside a KPI that represents the latency itself in the target space.
In this work, anomaly detection is done leveraging autoencoders and variational autoencoders (they will be explained in section 5), as done in [9] and [6], respectively. They are models that process the points in the dataset in such a way that they can be identified as anomalies and non-anomalies through a comparison with a threshold. This means that an additional dataset is needed, similar to the first, but with all the entries labelled as anomalies or non-anomalies.
Probabilistic regression and probabilistic Bayesian regression are performed by implementing a probabilistic neural network and a probabilistic Bayesian neural network (deepened in section 4). A probabilistic neural network is a model able to reproduce a parametrized probability distribution of the target value, hence able to quantify the aleatoric uncertainty of the latency. A probabilistic Bayesian neural network provides a principled way to account for a secondary source of uncertainty: the epistemic uncertainty, also referred to as model uncertainty, i.e., the model predictive uncertainty caused by the limited amount of data. In order to avoid model misspecification, a rigorous assumption on the probability density function of the latency must be introduced.

3.1. Datasets

In this work, two datasets provided by TIM are employed. They contain measurements of various KPIs collected from different base stations in different hours of the day. The first one gathers two consecutive days of data from all the base stations in the province of Bologna, with granularity of an hour, resulting in a quite large number of samples (107000 samples). This first dataset is used for the instantaneous regression.
The second dataset contains data gathered in two other consecutive days, with a granularity of 15 minutes, instead of an hour but a coverage of only 25 base stations, resulting in a quite smaller amount of data (7200 samples). The peculiarity of this second dataset is represented by a concert that took place under the coverage of two of the base stations and lasted 4.5

hours. The samples falling in the time and geographical interval of the concert are labelled as anomalies[1] and the dataset is used for anomaly detection.
In both the datasets, the entries are filtered by considering only the traffic in the 1800 MHz band.

3.2. Available KPIs

Both the datasets contain the same 67 KPIs, most of them belonging to two categories:

- average channel conditions
- network load (uplink and downlink) at each communication layer (physical, MAC, RLC, PDCP, IP and RRC)

There are other KPIs describing the relative quantities of connections using each MIMO format and finally, two additional KPIs, namely the average downlink latency of a PDCP SDU (i.e., average time interval from a SDU available at the PDCP layer base station side to the same SDU being made available to the upper layer by the PDCP layer UE side) in case of QCI 1 and QCI 7. The latter KPI is chosen as the target KPI due to its effectiveness in approximating the IP downlink latency and its relevance to the prevailing nature of mobile radio network traffic, which predominantly consists of QCI7 traffic. All the other KPIs (excluding the one referring to PDCP latency in case of QCI1) together represent the feature space.
In order to simplify the models, a process of "feature selection" is implemented, resulting in a reduction of the 65 KPIs of the feature space to a maximum of 10. This selection is carefully performed while ensuring that the resulting feature space remains as representative as possible of the target variable.

3.3. KPI selection

In the process of selecting an appropriate set of KPIs, an initial machine learning-based approach is followed, utilizing the "permutation importance" algorithm from the scikit-learn library. This selection is then reviewed through a "knowledge-based" approach, leveraging insights from existing literature and established practices.

[1] A concert is not an anomaly, but because of the high and unusual network load it generates, it can be simulated as such.

3.3.1. Permutation importance algorithm

The Permutation Importance algorithm evaluates the impact of shuffling the values of a single feature on the model's performance metric. It works by calculating a baseline performance metric (e.g., R2 score) using the original dataset, then, the values of a particular feature are randomly shuffled, breaking the relationship between the feature and the target. The model's performance metric is recalculated using the shuffled dataset, measuring the drop in performance caused by the shuffled feature. A higher drop in the performance metric indicates that the feature is more important for the model.

This process is repeated for each feature, measuring the importance of each one in isolation. The measured importance indicators can be sorted at the end, in order to obtain a list of features from the most relevant to the least relevant. The fact that it is applied on a test set, makes the importance measures realistic and not conditioned by possible overfitting.

The algorithm is applied on the first dataset (divided in a training and a test set), i.e., the one employed for the instantaneous regression, but the resulting feature selection is transferred also to the dataset utilized for anomaly detection.

As indicated by the documentation, prior to implementing the aforementioned algorithm, the KPIs are partitioned into clusters using a clustering model based on Spearman's rank correlation. Subsequently, only one feature per cluster is utilized in the permutation importance algorithm. This step is necessary when dealing with datasets comprising numerous columns, as it is highly probable that some of these columns exhibit strong correlations with each other. Consequently, when a column within such a correlated set is permuted, the decrease in score, and thus the measured importance, would be lower than the actual importance, because the model would still benefit from another correlated feature.

The partitioning of the KPIs based on Spearman's rank correlation is meticulously performed to yield 34 clusters. By selecting a single KPI from each cluster, the resulting set of 34 KPIs ensures a satisfactory level of decorrelation. This decorrelation enables the application of the permutation importance algorithm with optimal effectiveness.

3.3.2. Application and outcome of the permutation importance algorithm

Two ML models are chosen for the application of the permutation importance algorithm, namely decision tree regressor and random forest regressor. The first one is among the most popular regression algorithms; it uses a tree-like model to predict the value of a continuous target variable by recursively partitioning the data into subsets based on the feature values. The second one is an ensemble model that combines a number of decision tree regressors for

an improved performance (500 trees are chosen in this case). The two models are trained, then the permutation importance algorithm is applied, obtaining the importance of each of the 34 features.

The last step is to assume that the other features in each of the 34 clusters have similar importance with respect to one another. This assumption allows to build, for each of the two models, an ordered list of all 65 features, from the most important to the least. Within Table 1, both lists are truncated to include only the first 9 entries. The lists exclude KPIs associated with MIMO formats due to their spurious correlations with latency. Interestingly, despite appearing among the top 9 KPIs identified by the permutation importance algorithm, their actual impact on latency is deemed minimal.

Table 1: Top 9 KPIs according to permutation importance algorithm (from the most to the least important)

Decision tree regressor	**Random forest regressor**
average IP scheduled throughput in DL, QCI7	average IP scheduled throughput in DL, QCI7
average physical resource usage per TTI in DL	average physical resource usage per TTI in DL
PDCP SDU Volume in DL	PDCP SDU Volume in DL
average number of RRC connected UEs	average PDCP cell throughput in DL
max number of RRC connected UEs	PDCP SDU volume in UL
average PDCP cell throughput in DL	average physical resource usage per TTI in UL
inter-frequency handover attempts	average number of RRC connected UEs
PDCP SDU volume in UL	max number of RRC connected UEs
average physical resource usage per TTI in UL	RLC PDU volume in DL

3.3.3. Final KPI selection

In the literature, it is widely recognized that the KPIs that primarily influence latency are those associated with network load and channel conditions [2], [9].

However, in both selections made using the permutation importance algorithm, the nine most significant KPIs pertain to network load, while the KPIs addressing channel conditions are regarded as comparatively less significant.

To encompass KPIs associated with both network load and channel conditions, a manual selection process is employed. However, while considering the substantial significance attributed to network load by the permutation importance algorithm, a greater emphasis is placed on selecting KPIs related to network load. Consequently, six KPIs pertaining to

network load and two KPIs pertaining to channel conditions are chosen. Moreover, the time instant is included as an additional KPI.

Table 2: Final KPI selection.

Time instant
Total number of RRC setup attempts
PDCP SDU volume in DL
average physical resource usage per TTI in DL
average physical resource usage per TTI in UL
average active UEs in DL
average RRC connected UE
average CQI
average SINR for PUSCH

3.4. Probability density function of the latency

Skocaj et. Al in [9] provide a comprehensive mathematical derivation of the probability distribution function characterizing latency. Specifically, their analysis concentrates on U-plane latency, which refers to the delay from a packet being available at the IP layer, UE or base station side, to the same packet being available at the IP layer, base station or UE side. Notably, the C-plane latency, involving the delay experienced by a UE transitioning from RRC-idle to RRC-connected state, is not taken into consideration. This omission arises from the fact that C-plane latency mostly depends on the random access procedure, which is a procedure where the MNO does not have many degrees of freedom to perform optimization. Additionally, the study concentrates on downlink grant-based scheduling, as it represents the majority of the generated traffic. They define an expression for the U-plane latency:

$$L = \tau_{radio} + \tau_{HARQ} + N * \left(\tau'_{radio} + \tau_{HARQ}\right), \qquad (1)$$

where τ_{radio} denotes the delay of an IP packet transmitted through the Uu interface during the initial transmission, encompassing scheduling and transmission delays. Similarly, τ'_{radio} represents the same delay but in the case of a retransmission following the reception of a negative acknowledgement. Additionally, τ_{HARQ} denotes the fixed delay introduced by the Hybrid Automatic Repeat Request (HARQ) mechanism, while N represents the number of retransmissions. It is determined that both τ_{radio} and τ'_{radio} follow a negative exponential distribution with rates λ_1 and λ_2, respectively. This is due to their composition of waiting and service time in an M/M/1 queueing system. Furthermore, the number of retransmissions, N, is

geometrically distributed with a success parameter of $1-BLER$ (Block Error Rate). Since τ_{HARQ} remains constant, it can be added to τ_{radio} and τ'_{radio}, resulting in τ_{tx} and τ_{rtx}, respectively. Both τ_{tx} and τ_{rtx} continue to exhibit negative exponential distributions with rates λ_1 and λ_2, respectively. Authors in [9] continue by rewriting (1) in this way:

$$L_2 = \tau_{tx} \cdot P_0 + \sum_{j=1}^{N_{max}} (\tau_{tx} + j \cdot \tau_{rtx}) \cdot P_j \qquad (2),$$

where P_j is the probability of having j retransmissions and N_{max} is the maximum number of retransmissions. The sum $[\tau_{tx} \cdot P_0 + \sum_{j=1}^{N_{max}} \tau_{tx} \cdot P_j]$ is equal to τ_{tx}, so (2) reduces to:

$$L_2 = \tau_{tx} + \tau_{rtx} \cdot P_1 + 2 * \tau_{rtx} \cdot P_2 + \cdots + o(P_n) \qquad (3),$$

where all the terms in the sum are negative exponentially distributed, with rate λ_1 for the first term and $\frac{j*\lambda_2}{P_j}$ for all the following terms. This results in a hypoexponential distribution $hexp(\lambda_1, \ldots, \lambda_N)$, i.e., a distribution that sums together negative exponential distributions having different rates.

4. Instantaneous regression of the latency

For both probabilistic regression and probabilistic Bayesian regression, the 80% of the entries in the first dataset is used as training set, the remaining 20% as test set.

4.4. Probabilistic Regression

A probabilistic neural network (PNN) is a neural network made of a cascade of dense hidden layers followed by a probabilistic layer. A dense layer, also known as a fully connected layer, is a type of neural network layer where every neuron is connected to every neuron in the previous layer, allowing for complex non-linear transformations in data.

For this task, the activation of each dense layer is set as a rectified linear unit, which is a common choice to overcome the well known problems of vanishing and exploding gradients.

The probabilistic output layer is a function producing a parametrized distribution. The parameters of this distribution are the inputs of the layer, so, before this output layer, an additional dense layer, having as many neurons as the parameters of the distribution, must be inserted.

As demonstrated in Section 3, the latency distribution conforms to a hypoexponential distribution. In the Tensorflow library, there exists a distribution known as gamma, which exhibits a shape resembling that of a hypoexponential distribution. However, it should be noted that the gamma distribution is the sum of negative exponential distributions with the same rate, in contrast to the hypoexponential distribution, which combines negative exponentials with varying rates. This characteristic renders gamma a particular case of the hypoexponential distribution.

The depth of the probabilistic network, i.e., the number of hidden layers, is set equal 3, specifically, two dense layers, each one containing 16 nodes, followed by a dense layer of two nodes that generates the parameters α and β for the probabilistic layer producing gamma. Such number of layers and nodes per layer is already sufficient, and increasing the number of layers or nodes per layer does not improve the performance further. The shape parameter α determines how many negative exponential distributions are summed together, while the rate parameter β determines the rate of each of the negative exponential distributions. Both of them are learned during training.

When the output is a probability density function, the training of the neural network requires a loss function that measures the discrepancy between the estimated distribution and the real distribution, hence a negative loglikelihood

$$NLL = -\log[p(y_{train}|x_{train}, \theta)]\,, \qquad (4)$$

where x_{train} and y_{train} represent features and target of the training set and θ represents the set of weights and biases of the network. During training, the model parametrized by the vector of weights θ determines the output distribution from the input x_{train}. The likelihood is the probability of obtaining the realization y_{train} by sampling from the computed output distribution. The logarithm of the likelihood is commonly used to simplify computations, especially when dealing with small probabilities. The minus sign is added to the loglikelihood to make it a loss function to be minimized.

At the end of the training process, the model is evaluated by computing the Mean Absolute Error (MAE)[2] and the NLL on both the training and the test set (table 3).

Table 3: Evaluation of the probabilistic neural network

	Mean Absolute Error	Negative Loglikelihood
Training set	15.060 [ms]	-2.021
Test set	15.272 [ms]	-1.992

To visualize the quality of the estimation in terms of median and confidence intervals, the first 500 entries of the test set are given as inputs to the PNN. In order to retrieve aleatoric confidence intervals, a Monte Carlo experiment (100 repetitions) is performed. After obtaining 100 values for the estimated latency for each of the 500 points, median and confidence interval are computed over the 100 values for each of the points (figure 1).

As noticeable, the ground truth value confidently lies within the predicted confidence intervals. However, MAE and NLL computed for the training are slightly lower than those calculated for the test set. The cause is explained in [8]: by choosing gamma as output layer, the real data distribution is mis-specified, i.e., the output layer generates a different distribution to that of the test set, whose distribution is a hypoexponential. The consequence is that the PNN identifies certain points as outliers, even if they are not outliers when considering a hypoexponential distribution, but rather outliers in relation to a gamma distribution. Predictions for such points exhibit higher MAE and NLL, resulting in higher MAE and NLL values for the overall test set compared to the training set. However, the disparity between the evaluation scores of the training and test sets is minimal, indicating that the approximation of the hypoexponential distribution with gamma is quite accurate.

[2] For each test point, MAE is calculated on a single realization of the output distribution.

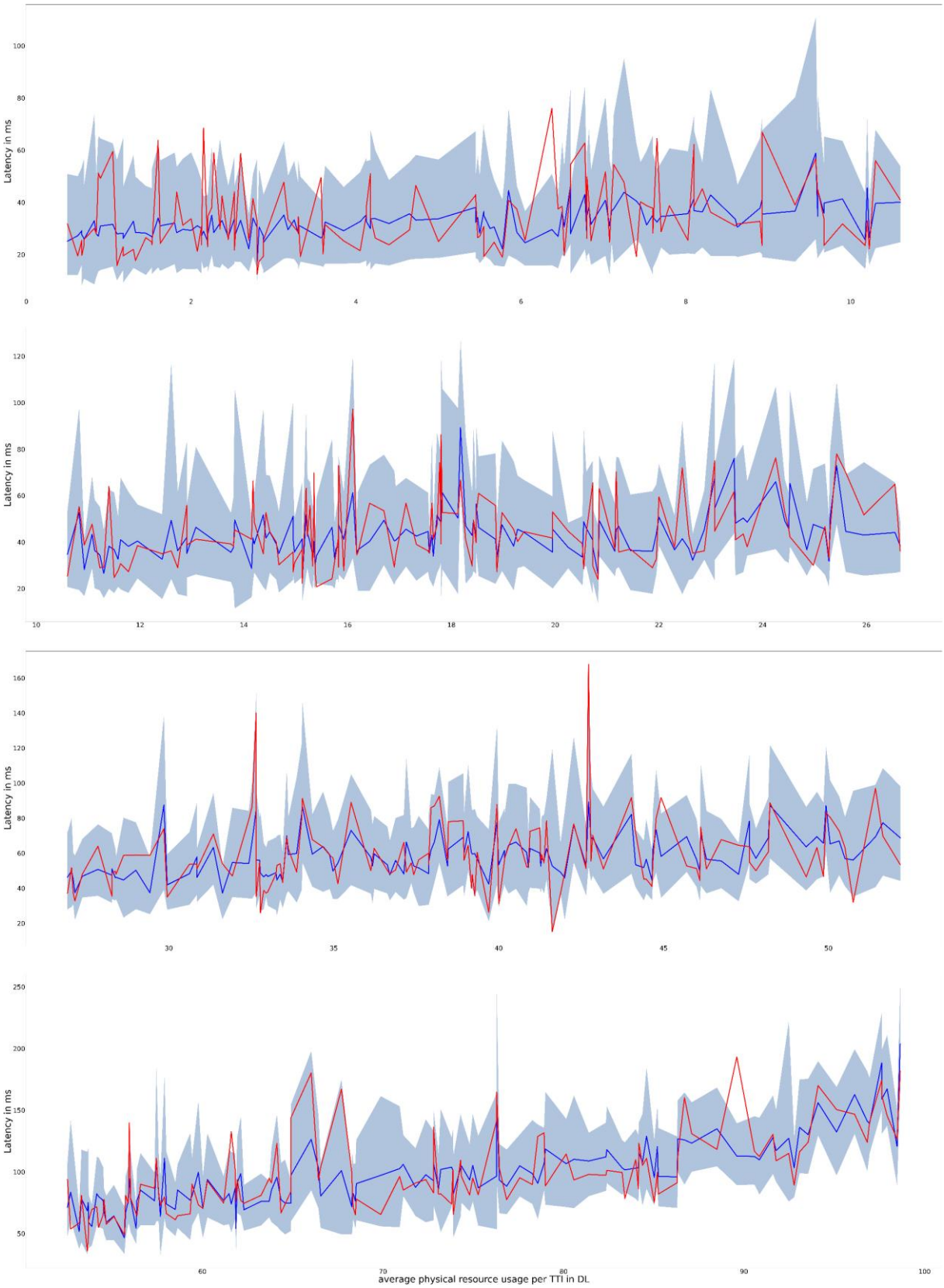

Figure 1: Plot for the probabilistic neural network: *The x-axis represents the average number of physical resources used per TTI in DL. This KPI is chosen for the plot, being one of the most representative.*

4.5. Probabilistic Bayesian Regression

The above descripted probabilistic neural network estimates the uncertainty given by the latency, that is an aleatoric process. As mentioned in section 3, there is another type of uncertainty called epistemic uncertainty, that refers to the uncertainty arising from a lack of knowledge in predicting a given target value. Unlike aleatoric uncertainty, epistemic uncertainty can be reduced by collecting more data. However, as highlighted in [8], in wireless communications obtaining abundant data is not always feasible, especially at the physical layer, in which the stationary intervals for data collection are short. Thus, the solution in this case is not the reduction of the epistemic uncertainty but its estimation.

In a Bayesian neural network [7], the epistemic uncertainty is reproduced by treating the set of weights and biases θ of the neural network not as a set of deterministic values but as distributed according to a multivariate posterior distribution $p(\theta|x_{train}, y_{train})$. Whenever the model is tested, a distinct set of weights and biases is randomly drawn from the posterior distribution. This is formally equivalent to a weighted ensemble model, where the probability of each given weight is modeled by the posterior distribution and updated by means of Bayesian inference.

During training, the posterior distribution is updated using Bayes theorem:

$$p(\theta|x_{train}, y_{train}) = \frac{p(y_{train}|x_{train}, \theta) * p(\theta)}{p(y_{train}|x_{train})} \qquad (5)$$

The prior distribution $p(\theta)$ is arbitrary and the denominator is a known distribution that does not depend on the model, hence, the variability of the network weights and biases, mostly depends on how much $p(y_{train}|x_{train}, \theta)$ is a skewed distribution. Intuitively, a bigger and more diverse training set (x_{train}, y_{train}) can be reproduced with good approximation by fewer neural network configurations. In that case $p(y_{train}|x_{train}, \theta)$ will be skewer, and consequently, according to Bayes theorem, the output will be less variable. This is the way the epistemic uncertainty is reproduced.

By maintaining a similar architecture to that of the probabilistic neural network, a probabilistic Bayesian neural network (PBNN) is implemented. It is a model that captures both epistemic and aleatoric uncertainty. The output probabilistic layer reproducing the gamma distribution and the number of hidden layers and nodes per hidden layer remains the same of the PNN, while the type used for the hidden layers changes from “dense” to “dense variational”, following an approach called “variational inference”.

Variational inference allows the actual implementation of the Bayes theorem in a neural network by approximating the multivariate posterior distribution $p(\theta|x_{train}, y_{train})$ (that can

be a very complex distribution) with a set $q_\lambda(\theta)$ of parametrized distributions, one per network weight or bias. The parametrized distributions are usually chosen as independent Gaussian distributions, each parametrized by a mean and a variance, so the set of parameters λ to be learnt is only twice the number of the network weights. As showed in [7], the loss function that must be minimized is the KL divergence between $q_\lambda(\theta)$ and the real posterior distribution $p(\theta|x_{train}, y_{train})$, hence,

$$KL[\, q_\lambda(\theta) \| \, p(\theta|x_{train}, y_{train})] \qquad (6).$$

By applying the definition of the KL divergence and then substituting the conditional probabilities with the product between joint and marginal distribution, (6) becomes:

$$KL[\, q_\lambda(\theta) \| \, p(\theta|x_{train}, y_{train})] = \int q_\lambda(\theta) log \frac{q_\lambda(\theta)}{p(\theta|x_{train}, y_{train})} d\theta$$

$$= \int q_\lambda(\theta) log \frac{q_\lambda(\theta)}{p(\theta, y_{train}|x_{train})} d\theta + \int q_\lambda(\theta) \log[p(y_{train}|x_{train})]\, d\theta \,.$$

The term $\log[p(y_{train}|x_{train})]$ does not depend on θ, so it can be taken outside the integral, thus, the second integral becomes $\int q_\lambda(\theta)\, d\theta$, that is equal 1. At this point, the first integral is the only term of the loss that can be minimized:

$$\lambda_{PBNN} = argmin\left\{ \int q_\lambda(\theta) log \frac{q_\lambda(\theta)}{p(\theta, y_{train}|x_{train})} d\theta \right\}.$$

By substituting the joint probability with the product between conditional and marginal probabilities, the above expression becomes:

$$\lambda_{PBNN} = argmin\left\{ \int q_\lambda(\theta) log \frac{q_\lambda(\theta)}{p(y_{train}|x_{train}, \theta) p(\theta)} d\theta \right\}$$

$$= argmin\left\{ \int q_\lambda(\theta) log \frac{q_\lambda(\theta)}{p(\theta)} d\theta - \int q_\lambda(\theta) log[p(y_{train}|x_{train}, \theta)] d\theta \right\}.$$

Hence, the final expression for the loss used in a PBNN is:

$$\lambda_{PBNN} = argmin\{KL[\, q_\lambda(\theta) \| \, p(\theta)] - E_{\theta \sim q_\lambda}[log[p(y_{train}|x_{train}, \theta)]]\} \qquad (7)$$

The second term is the negative loglikelihood, seen in section 4.1. The first term, called *equalizer*, is added by the dense variational layers, and it forces the parametrized approximated posterior distribution to not diverge too much from the imposed prior distribution. A set of independent normal distributions is chosen as the prior distribution, as it is common choice.

The PBNN is trained, and then evaluated on both the training and the test set with MAE and NLL (table 4), like done for the probabilistic neural network.

Table 4: Evaluation of the probabilistic Bayesian neural network.

	Mean Absolute Error	Negative Loglikelihood
Training set	15.730 [ms]	-1.982
Test set	15.850 [ms]	-1.951

Median and confidence interval of the PBNN are computed in the same way as done with the PNN, so by repeating each prediction 100 times.

In Figure 2, the estimated medians and confidence intervals of the PNN and the PBNN are superimposed. The median values and confidence intervals between the two models exhibit negligible differences, indicating that both the PBNN and the PNN converge equally well. Focusing on the evaluation results, the MAEs and NLLs are higher than in the only probabilistic case. This is due to the estimation of the epistemic uncertainty, which, despite the model convergence, remains non-zero for latency values that are underrepresented in the dataset. Such points contribute to the deterioration of the MAEs and NLLs.

Just like the PNN, the mis-specification of the output distribution results in slightly higher MAE and NLL for the test set compared to the training set. In the case of the PBNN, the points that are wrongly identified as outliers by the mis-specified distribution, are also predicted with a very high epistemic uncertainty, because the model is not able to generalize on them. In [8] this aspect is identified as one of the weaknesses of Bayesian learning. However, considering that the suboptimality of the gamma distribution may impact the epistemic uncertainty, the confidence intervals computed by the PBNN using gamma may only be larger than those computed by a PBNN using a hypoexponential distribution. This could be advantageous when estimating the epistemic uncertainty of the latency using a conservative approach.

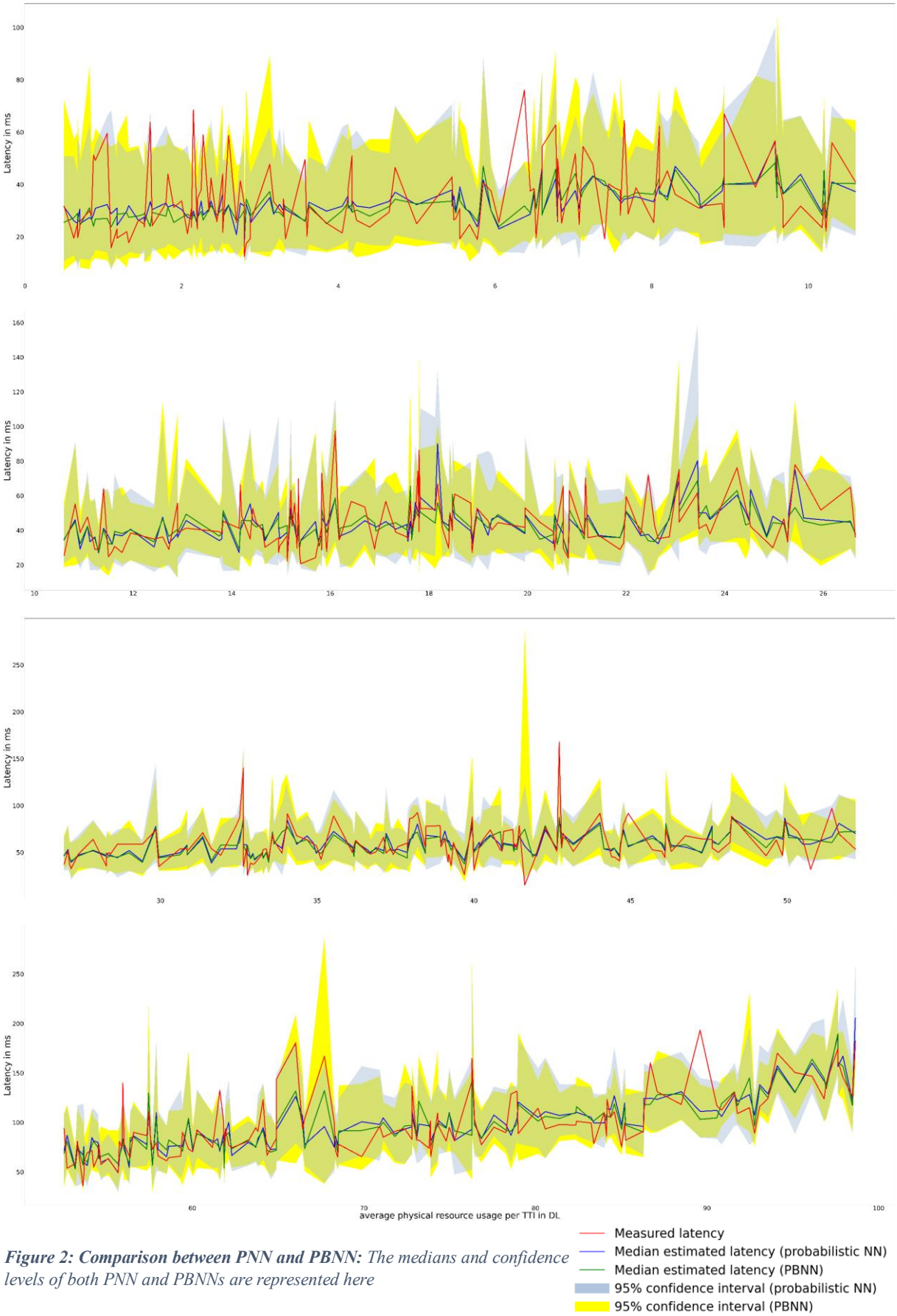

Figure 2: Comparison between PNN and PBNN: *The medians and confidence levels of both PNN and PBNNs are represented here*

5. Anomaly detection

5.4. The anomaly detection task

Anomaly detection refers to the identification of anomalies among the target points. This is done through a model parametrized with a set of weights and biases θ. When the model is tested on a point (*x,y*), a probability density function $f(x, y|\theta)$ is computed [9]. If the probability density function is lower than a likelihood threshold, the point is identified as an anomaly.

There are two alternative approaches. According to the first one, the probability density function is the likelihood $p(y|x, \theta)$, hence the model to be used is a probabilistic or a probabilistic Bayesian neural network. According to the second one, the probability density function is a joint distribution $p(y, x|\ \theta)$ learnt by the model. The models that follow the second approach are not probabilistic networks, but deterministic networks that try to reproduce the (*x,y*) points according to the learnt joint distribution. In this case, the threshold cannot be directly set on $p(y, x|\ \theta)$, but only on the output reconstruction error, i.e., the error in reproducing (*x,y*), hence, those points reconstructed with error higher than the threshold are labelled as anomalies. However, the output depends on the learnt joint distribution, so this is equivalent to set the threshold on $p(y, x|\ \theta)$ and considering points falling below the threshold as anomalies.

In this work the second approach is adopted, and some autoencoders and variational autoencoders are compared to each other.

5.5. Autoencoders

An autoencoder (AE) is composed of two parts: an encoder, that compresses input data into a lower-dimensional representation, and a decoder, that usually has a specular architecture with respect to the encoder and reconstructs the original input from the compressed representation. The final layer of the encoder produces the input for the decoder. This layer, called latent layer, generates a latent space, a representation of the autoencoder input. The latent layer typically has less dimensions than the input layer, this is why the latent space is a compressed representation of the input space.

As anticipated in section 3, datasets containing points labelled as anomalies and non-anomalies are used for anomaly detection. The training set contains only non-anomalies, so the model learns only the distribution of the non-anomalies and encodes it in the latent space. Therefore, after training, if tested on a non-anomaly, the model reproduces it with a low

reconstruction error, but if tested on an anomaly, the model is not able to rightly reproduce it and the resulting reconstruction error will be higher.

The metric chosen to measure the reconstruction error is the Mean Squared Error (MSE):

$$MSE\left[\left(x^{estimated}, y^{estimated}\right), (x, y)\right] = E\left\{\left[\left(x^{estimated}, y^{estimated}\right) - (x, y)\right]^2\right\}. \quad (8)$$

MSE is used also as loss function for training the autoencoder.

5.6. Evaluation phase

For the evaluation, the relative quantities of true positives (TP), true negatives (TN), false positives (FP) and false negatives (FN) must be considered. In the used dataset (as generally happens when dealing with anomalies), the non-anomalies are much more than the anomalies, and moreover, the anomalies span a much larger reconstruction error interval. The only way to find an optimal threshold is not to arbitrarily set its value, but to leave it as a parameter to be tuned after training. This is done by creating a custom score to be minimized after training, the expression is the following:

$$custom\ score = \frac{\frac{TN}{TN + FP} + \beta * \frac{TP}{TP + FN}}{1 + \beta} \quad (9),$$

the first term at the numerator is the *specificity score*, and the term that multiplies β is the *recall score*. These two terms are chosen to make the difference between the quantities of positives and negatives irrelevant. Their average is weighted on a parameter β, that acts as a scaling constant against the difference in the intervals of the reconstruction errors of the positives and the negatives. β must be tuned together with the threshold, to minimize the custom score.

5.7. Variational autoencoders

A variational autoencoder (VAE) is an AE with a variational latent space, learnt with variational inference [10]. Thanks to the Bayesian theorem, a VAE exhibits greater efficacy in capturing the data distribution compared to an AE. With VAEs, labelling anomalies and non-anomalies in advance is unnecessary, allowing the anomaly detection task to be reformulated as an unsupervised problem [6]. In this reformulated approach, anomalies are detected by identifying their dissimilarity in relation to the learned distribution.

However, the VAEs employed in this study are utilized with a labeled dataset, thereby retaining a supervised nature for the problem. This approach ensures a fair comparison between AEs and VAEs built with similar architectures.

Unlike PBNNs, in VAEs variational inference is not implemented by using a dense variational layer but working "from scratch". In practice, a prior multivariate distribution is defined, consisting of a collection of untrainable, independent normal distributions. The distribution $q_\lambda(\theta)$, that approximates the real posterior distribution of the latent space, is the latent layer, and it is directly derived from the prior, by adding the mean and multiplying for the variance. Mean and variance are the multidimensional outputs of two additional dense layers, both having as input the last hidden layer of the encoder.

The network is trained by finding the minimum λ for (7). The second term of the equation is now substituted with the MSE, because the model is no more probabilistic but deterministic[3]. The equation (7) is further modified by incorporating a scaling factor w to multiply the regularizer. This modification provides the ability to control and decrease the variability of the latent space, as it enforces proximity to the prior distribution as closely as possible. In fact, the objective of the VAE is not the representation of the epistemic uncertainty, but the reproduction of the data distribution. Here is the expression of the loss used for training VAEs:

$$\lambda_{VAE} = argmin\{w * KL[\, q_\lambda(\theta) \| p(\theta)] - MSE\left[\left(x_{train}^{estimated}, y_{train}^{estimated}\right), (x, y)\right]\} \quad (10)$$

When tested, each variational autoencoder undergoes 100 Monte Carlo experiments producing 100 reconstruction errors for each point of the test set. Subsequently, evaluations are conducted by minimizing the custom scores, similar to the approach used for standard autoencoders.

5.8. Comparison between different types of autoencoders and variational autoencoders

In this section, two distinct architectural configurations are tested. The first configuration comprises an encoder and decoder, each composed of two hidden layers consisting of 16 nodes. The latent layer in this configuration has two dimensions. In contrast, the second configuration uses four hidden layers, with each layer containing 16 nodes, for both the

[3] This approximation perfectly works, in fact the reconstruction error is itself related to a discrepancy between truth and estimation, conditioned by the network weights and biases, like it is for the NLL.

encoder and decoder. Additionally, the latent dimension in the second configuration is set to 16.

Table 5: Comparison between different types of AEs and VAEs

MODEL	Tuned threshold	Tuned β	Training accuracy	Test accuracy	Test F1 score	TP	TN	FP	FN	Test custom score
First architecture (4 hidden layers, latent space dimension = 2)										
AE	0.556	0.5	0.963	0.966	0.735	36	693	24	2	0.960
VAE (w=1)	3.883	0.5	0.990	0.986	0.870	3509	70946	754	291	0.967
VAE (w=2)	9.886	0.5	0.998	0.997	0.973	3600	71700	0	200	0.982
Second architecture (8 hidden layers, latent space dimension = 16)										
AE	0.450	0.5	1	0.984	0.813	26	717	0	12	0.895
VAE (w=1)	6.067	0.5	0.996	0.994	0.937	3565	71456	244	235	0.977
VAE (w=2)	9.886	0.5	0.999	0.997	0.973	3600	71700	0	200	0.982

For both the architectures, three networks are tested: an AE, a VAE trained with scaling factor from (10) $w = 1$, and a VAE trained with $w = 2$. The results are synthetized in the table 5. Remembering that both the threshold and β are tuned by minimizing the custom score after training, all the models appear to prefer $\beta = 0.5$, thus penalizing the recall score in the (9). This means that all the networks solve the problem of the higher concentration of negatives by penalizing the positives. The consequence can be seen in figure 3: all the networks tune a high threshold on the MSE, minimizing the number of false positives. This may be useful in a typical scenario where anomalies are rare events and thus too many false alarms would negatively impact the QoS.

When comparing the two architectures, it's hard to determine a clear winner. The first architecture appears to outperform the others for the AE, as indicated by a higher custom score. However, for the VAE, the second architecture seems to be more effective.

Another observation is that the custom score is fairer than both the accuracy and the F1 scores; for example, according to the F1 score, the AE with the second architecture would perform better than the other AE, while the relative quantities of FP and FN say the opposite, giving credit to the custom score.

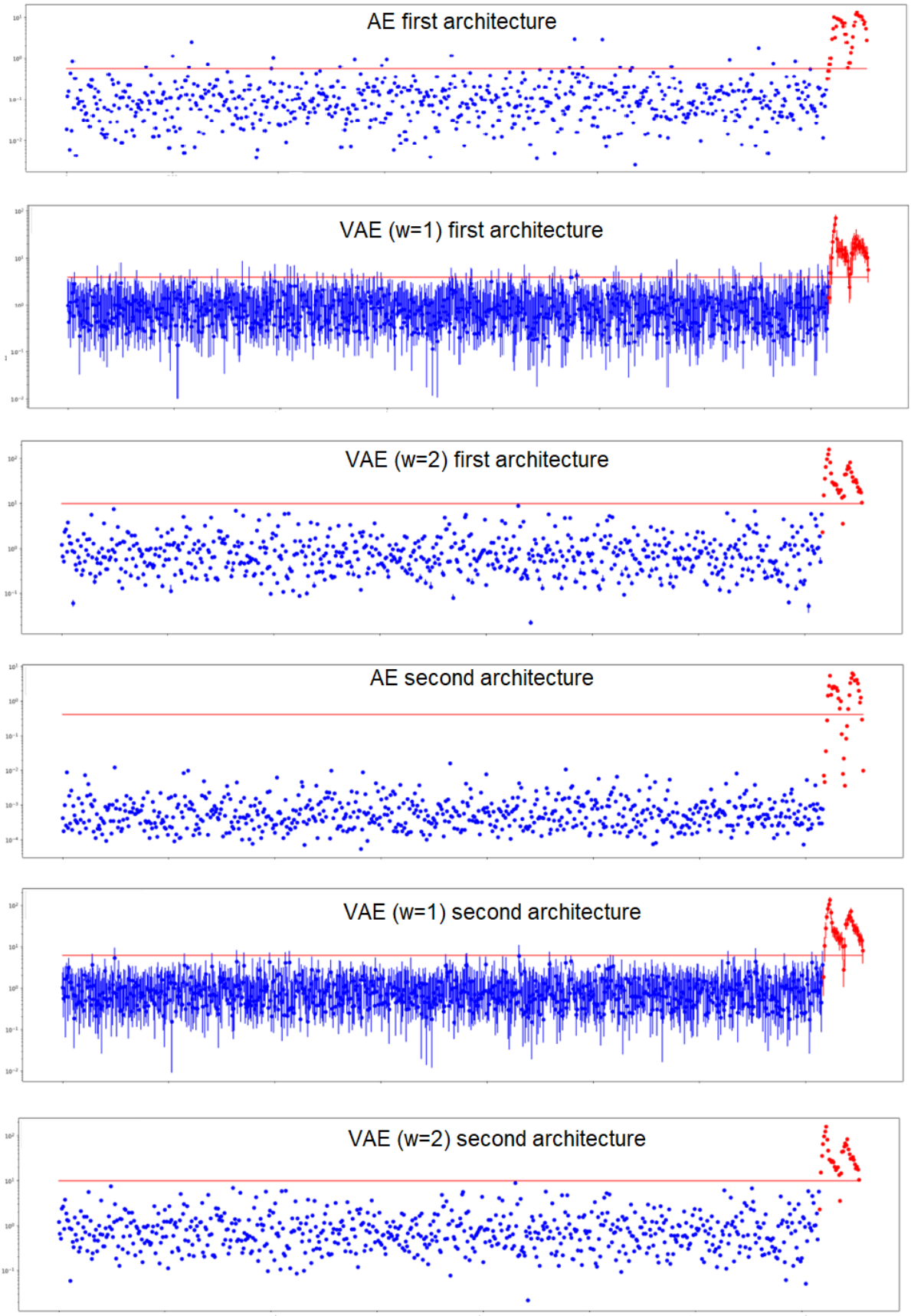

Figure 3: Anomaly detection: *Mean Squared Errors (logarithmic scale) for all the points of the test set (VAEs are tested 100 times, so, for each point, 100 MSEs are traced as a vertical line that traverses a point representing their mean). The red points above the threshold are TP, those falling below are FN. The blue points above the threshold are FP, those falling below are TN.*

Finally, the most important observation is about the better performances of VAEs with respect

to AEs, according to every scoring functions. The reconstruction errors generated by VAEs are much higher, but the separation between anomalies and non-anomalies is clearer. This confirms that the latent distribution of the non-anomalies learnt with variational inference is more realistic than the one learnt by the AEs, as stated in [6], and thus, anomalies are better recognized. In particular, the VAE trained with w=2 performs better than the one weighted with w=1. The former has a very low variability (for almost all the test points, the 100 reconstruction errors of the Montecarlo experiment coincide), so it is forced to learn a latent distribution that is the most similar possible to the prior, and such approximation improves the detection.

6. Conclusions

This study started with a machine learning-based KPI selection process, which revealed that the KPIs related to the network traffic have a greater influence on the latency compared to the KPIs related to the channel conditions.

Probabilistic regression was employed using a probabilistic neural network generating a parametrized gamma distribution as the output. Remarkably, gamma proved to be a reliable approximation of the hypoexponential distribution of latency, as the mis-specification of such output distribution had minimal impact on the test scores.

The implementation of probabilistic Bayesian regression involved transforming the dense layers of the PNN into dense variational layers. The convergence of the resulting PBNN closely resembled that of the PNN, albeit with slightly higher MAE and NLL attributed to certain data points exhibiting epistemic uncertainty. Notably, this uncertainty is occasionally overestimated due to the mis-specification introduced by the gamma distribution. However, only Bayesian learning can offer an estimation, albeit sometimes conservative, of the credibility of a result, rendering the representation of PBNNs more realistic.

For instance, a base station estimating latency and making decisions based on such estimates can weigh its choices according to the levels of aleatoric and epistemic uncertainty.

Regarding anomaly detection, six models were examined: an autoencoder and two variational autoencoders, each with two different architectures. VAEs definitely appear to outperform AEs. The improvement achieved by incorporating variability into the latent layer is significantly greater than that achieved by adjusting the number of layers and nodes per hidden layer. In particular, the VAE trained with a loss regularized by a factor $w = 2$ is the one giving the best results, regardless of the number of hidden layers and nodes per hidden layers.

7. Bibliography

[1] A. Zappone, M. Di Renzo and M. Debbah, "Wireless Networks Design in the Era of Deep Learning: Model-Based, AI-Based, or Both?", in IEEE Transactions on Communications, vol. 67, no. 10, pp. 7331-7376, Oct. 2019, doi: 10.1109/TCOMM.2019.2924010.

[2] M. Boban et al., "Predictive Quality of Service: The Next Frontier for Fully Autonomous Systems," IEEE Network, 2021.

[3] D.C. Moreira et al., "QoS Predictability in V2X Communication with Machine Learning," in Proc. 91st Annual Int. Veh. Technol. Conf., 2020.

[4] Weiwei Jiang, "Cellular traffic prediction with machine learning: A survey", Expert Systems with Applications, Volume 201, 2022

[5] A. Garg, W. Zhang, J. Samaran, R. Savitha and C. -S. Foo, "An Evaluation of Anomaly Detection and Diagnosis in Multivariate Time Series", in IEEE Transactions on Neural Networks and Learning Systems, vol. 33, no. 6, pp. 2508-2517, June 2022, doi: 10.1109/TNNLS.2021.3105827.

[6] Haowen Xu, Wenxiao Chen, Nengwen Zhao, Zeyan Li, Jiahao Bu, Zhihan Li, Ying Liu, Youjian Zhao, Dan Pei, Yang Feng, Jie Chen, Zhaogang Wang, and Honglin Qiao. 2018. "Unsupervised Anomaly Detection via Variational Auto-Encoder for Seasonal KPIs in Web Applications", in Proceedings of the 2018 World Wide Web Conference (WWW '18). International World Wide Web Conferences Steering Committee, Republic and Canton of Geneva, CHE, 187–196.

[7] Oliver Dürr, Beate Sick, with Elvis Murina "Probabilistic deep learning"

[8] M. Zecchin, S. Park, O. Simeone, M. Kountouris and D. Gesbert, "Robust Bayesian Learning for Reliable Wireless AI: Framework and Applications", in IEEE Transactions on Cognitive Communications and Networking, doi: 10.1109/TCCN.2023.3261300.

[9] M. Skocaj, F. Conserva, N. Sarcone Grande, A. Orsi, D. Micheli, G. Ghinamo, S. Bizzarri, and R. Verdone, "Data-driven Predictive Latency for 5G: a Theoretical and Experimental Analysis using Network Measurements"

[10] Kingma, Diederik P., and Max Welling. "Auto-encoding variational bayes." arXiv preprint arXiv:1312.6114 (2013)